

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



TRABAJO FIN DE MÁSTER

Detección de bots basada en modelado de interacción táctil y sensores inerciales

Máster Universitario en Ingeniería de Telecomunicación

Autor: Lo Russo Pico, Ágata

Tutor: Morales Moreno, Aythami

Índice general

1. Introducción	1
1.1. Motivación y objetivos del proyecto	2
1.2. Estado del arte	3
1.3. Metodología y plan de trabajo	4
2. Estudio de señales derivadas de la interacción Humano-Máquina con dispositivos móviles	6
2.1. Descripción de la base de datos	6
2.2. Análisis de los datos de sensores	8
2.2.1. Secuencias de interacción táctil	9
2.2.2. Secuencias de acelerómetro	11
2.2.3. Análisis estadístico de los datos disponibles de cada sensor . . .	13
2.2.4. Extracción de características de los datos: Velocidad instantánea	16
2.3. Estudio de la correlación entre secuencias de interacción táctil y acelerómetro	18
2.3.1. Estudio de la correlación mediante parámetros estadísticos . . .	18
2.3.2. Estudio de la correlación mediante el empleo de aprendizaje supervisado	20
2.3.2.1. Resultados obtenidos	23

3. Redes Neuronales Generativas (GAN)	25
3.1. Definición	25
3.2. Hiperparámetros	27
3.3. Utilidad de las redes GAN en el contexto del proyecto	29
4. Síntesis de secuencias temporales asociadas a la interacción con dispositivos móviles basada en GAN	31
4.1. Diseño y entrenamiento de una red GAN para generación y discriminación de secuencias de interacción táctil	32
4.1.1. Resultados obtenidos	35
4.2. Diseño y entrenamiento de una red GAN para generación y discriminación de secuencias de acelerómetro	37
4.2.1. Resultados obtenidos	39
4.3. Verificación de las secuencias generadas mediante el empleo de redes neuronales	41
4.3.1. Clasificador de secuencias reales y ruido gaussiano	42
4.3.1.1. Verificación de resultados de interacción táctil	42
4.3.1.2. Verificación de resultados de acelerómetro	44
4.4. Verificación de la eficiencia del discriminador en la detección de bots y propuesta de modelos alternativos	45
4.4.0.1. Verificación de resultados de interacción táctil	46
4.4.0.2. Verificación de resultados de acelerómetro	47
4.4.1. Clasificador de secuencias generadas y secuencias reales	48

4.4.1.1.	Verificación de resultados de interacción táctil	49
4.4.1.2.	Verificación de resultados de aceleración	51
4.5.	Diseño y entrenamiento de una red GAN para generación y discriminación de secuencias de interacción táctil y acelerómetro simultáneamente	52
4.5.1.	Modelo de generador secuencial	53
4.5.2.	Modelo de generador funcional	55
4.5.3.	Conclusiones	57
5.	Desarrollo de la plataforma de captura de datos BeCaptcha Web	59
5.1.	Obtención de datos de sensores vía web	60
5.2.	Obtención de datos procedentes de la interacción Humano-Dispositivo .	61
5.2.1.	Tecnologías empleadas en el desarrollo de la web	62
5.2.2.	Detección del navegador y dispositivo	64
5.2.3.	Captura de sensores de los dispositivos móviles	66
5.2.3.1.	Acelerómetro	67
5.2.3.2.	Giroscopio	68
5.2.3.3.	Magnetómetro	69
5.2.3.4.	Orientación del dispositivo	69
5.2.3.5.	Otros sensores implementados	72
5.2.4.	Captura de teclado, ratón y pantallas táctiles	72
5.2.4.1.	Teclado	72

5.2.4.2. Ratón (Mouse)	73
5.2.4.3. Pantalla táctil (Touchscreen)	73
5.2.5. Implementación de Google reCAPTCHA v3	75
5.2.6. Implementación de base de datos en tiempo real con Google Firebase	76
5.2.6.1. Procesamiento de la base de datos con Python	79
5.3. Rendimiento de la solución implementada y limitaciones conocidas . . .	79
6. Desarrollo de un demostrador web con Flask	82
6.1. Descripción de la aplicación web	82
6.2. Conclusiones	85
6.3. Subida de la aplicación a Github y Azure Web Services	86
7. Conclusiones y líneas futuras	90
7.1. Conclusiones	90
7.2. Líneas futuras de investigación	91
Bibliografía	93

Índice de figuras

2.1. Descripción de todos los sensores disponibles en la base de datos HuMldb. E = Adquisición basada en evento. Fuente: [1]	7
2.2. Interfaz de las ocho tareas de la aplicación de captura de los datos de HuMldb. Fuente: [1]	8
2.3. Estructura de carpetas de la base de datos HuMldb. Fuente: [1]	9
2.4. Ejemplos de movimientos de interacción táctil (deslizar el dedo hacia la derecha), disponibles en la base de datos de HuMldb, tras el procesamiento.	11
2.5. Sistema de coordenadas relativo a un dispositivo móvil, en sistemas Android.	12
2.6. Ejemplos de vectores de aceleración lineal producidos durante una interacción táctil hacia la derecha. Se representa la aceleración normalizada frente al número de muestra en cada instante.	13
2.7. Histogramas del número de muestras por movimiento, (a) Interacción táctil, (b) Acelerómetro Lineal.	14
2.8. Histogramas de la distribución del tiempo entre muestras, (a) Interacción táctil, (b) Acelerómetro Lineal.	16
2.9. Histogramas de la frecuencia de muestreo, (a) Interacción táctil, (b) Acelerómetro Lineal.	16
2.10. Datos de posición y velocidad instantánea para dos movimientos asociados de interacción táctil y acelerómetro. El eje horizontal representa el número de muestra.	17

2.11. Histograma de los coeficientes de correlación entre las velocidades instantáneas de los sensores de interacción táctil y acelerómetro.	19
2.12. Capas de la red neuronal empleada para el estudio de la correlación entre sensores.	22
2.13. Resultados de entrenamiento del clasificador para tres variaciones de los datos de entrada.	23
3.1. Arquitectura propuesta para entrenar una red GAN. Fuente: [2]	25
3.2. Funciones de activación comúnmente empleadas. (a) ReLu (b) Sigmoide (c) tanh	29
4.1. Pérdidas de la red GAN por época.	36
4.2. Ejemplos de secuencias de interacción táctil reales frente a las generadas, obtenidas con el modelo descrito.	37
4.3. Distribución de los datos reales y generados por característica. Interacción táctil. Izquierda: Valores de X. Derecha: Valores de Y	38
4.4. Pérdidas de la red GAN por época.	39
4.5. Ejemplo de secuencia de aceleración real frente a una generada, obtenida con el modelo descrito.	40
4.6. Distribución de los datos reales y generados por característica. Acelerómetro. Izquierda: Valores de X. Centro: Valores de Y, Derecha: Valores de Z.	41
4.7. Resultados de <i>accuracy</i> y <i>loss</i> para el clasificador binario entre secuencias reales y ruido aleatorio gaussiano, entrenado con secuencias de interacción táctil.	43

4.8. Resultados de clasificación. (a) <i>Scores</i> obtenidos para cada secuencia tras la clasificación. (b) Histogramas con los valores de los scores obtenidos para cada tipo de secuencia.	43
4.9. Resultados de <i>accuracy</i> y <i>loss</i> para el clasificador binario entre secuencias reales y ruido aleatorio gaussiano, entrenado con secuencias de aceleración.	44
4.10. Resultados de clasificación. (a) <i>Scores</i> obtenidos para cada secuencia tras la clasificación. (b) Histogramas con los valores de los <i>scores</i> obtenidos para cada tipo de secuencia.	45
4.11. Distribución de <i>scores</i> obtenidos tras clasificar 5000 secuencias con el discriminador de secuencias de interacción táctil.	46
4.12. Distribución de <i>scores</i> obtenidos tras clasificar 5000 secuencias con el discriminador de secuencias de aceleración.	47
4.13. Resultados de <i>accuracy</i> y <i>loss</i> para el clasificador binario entre secuencias reales y generadas con la GAN, entrenado con secuencias de interacción táctil.	49
4.14. Distribución de <i>scores</i> obtenidos tras clasificar 5000 secuencias con el clasificador (Real frente a Generado con GAN) de secuencias de interacción táctil.	50
4.15. Resultados de <i>accuracy</i> y <i>loss</i> para el clasificador binario entre secuencias reales y generadas (GAN), entrenado con secuencias de aceleración. . .	51
4.16. Distribución de <i>scores</i> obtenidos tras clasificar 5000 secuencias con el clasificador (Real frente a Generado con GAN) de secuencias de aceleración. (a) Todas las secuencias (b) Zoom de los scores de secuencias de ruido aleatorio gaussiano.	51
4.17. Modelos secuenciales probados para el generador de la GAN propuesta, para el entrenamiento de dos sensores simultáneamente.	54

4.18. Modelo funcional propuesto para el generador de la red GAN. Camino izquierdo: Entrenamiento de interacción táctil. Camino derecho: Entrenamiento de aceleración.	56
4.19. Ejemplo de secuencias obtenidas con la GAN con modelo de generador funcional. (a) Interacción táctil (b) Aceleración	57
5.1. Plataforma de captura de datos vía web vista desde Google Chrome (izquierda), junto con la consola del navegador (derecha). URL: becaptcha.10x.es	63
5.2. Sistema de coordenadas local del sensor de acelerómetro. Fuente: [13] .	67
5.3. Sistema de coordenadas local del sensor de giroscopio. Fuente: [14] . . .	68
5.4. Sistema de coordenadas local del sensor de magnetómetro. Fuente: [17]	69
5.5. Marco de coordenadas del dispositivo. Fuente: [15]	71
5.6. Movimientos asociados a los ángulos <i>alpha</i> , <i>beta</i> y <i>gamma</i> en un dispositivo móvil. Fuente: [15]	71
5.7. Respuesta en formato JSON que devuelve la petición de reCAPTCHA v3.	76
5.8. Ejemplo de dos conexiones realizadas a la página web, desde diferentes dispositivos, visto desde la herramienta de gestión de Google Realtime Database.	78
5.9. Aspecto de la base de datos obtenida tras el postprocesado con Python.	79
6.1. Página principal de la aplicación desarrollada: BeCaptchaWeb.	83
6.2. Cuadro para la captura de secuencias de interacción táctil, junto con la secuencia de aceleración asociada.	84

6.3.	Resultados tras generar una secuencia de interacción táctil a partir de ruido aleatorio. (a) Secuencia generada y resultados de detección de bots.	
	(b) Evaluación de los resultados del generador e información adicional.	85
6.4.	Configuración de la aplicación web desde Azure.	87
6.5.	Repositorio de GitHub con la aplicación del proyecto. URL: https://github.com/alrtfm/becapthaweb	88
6.6.	Despliegue de la aplicación en Azure Web Services desde el repositorio GitHub	89

Índice de tablas

2.1. Características de los datos analizados.	14
2.2. Percentiles obtenidos para el numero de muestras por cada movimiento del sensor	15
4.1. Matriz de confusión para para el clasificador de secuencias de interacción táctil, con umbral 0.3, para un total de 5000 muestras por clase.	47
4.2. Matriz de confusión para el clasificador de secuencias de aceleración, con umbral 0.3, para un total de 5000 muestras por clase.	48
4.3. Matriz de confusión para el clasificador de secuencias de aceleración, con umbral 0.6, para un total de 5000 muestras por clase.	48
4.4. Matriz de confusión para el clasificador de secuencias reales frente a generadas (GAN) de interacción táctil, con umbral 0.55, para un total de 5000 muestras por clase.	50
4.5. Matriz de confusión para el clasificador de secuencias reales frente a generadas (GAN) de aceleración, con umbral 0.5, para un total de 5000 muestras por clase.	52
5.1. Porcentajes de uso de los navegadores web en el mundo, para diferentes plataformas.	61
5.2. Compatibilidad de las APIs implementadas con los navegadores más utilizados a nivel mundial, para navegadores convencionales y navegadores para dispositivos móviles.	80

1. Introducción

El presente trabajo fin de Máster se desarrolla en el contexto del reconocimiento biométrico. Concretamente, en el estudio de la interacción humano-máquina con dispositivos móviles y la detección de bots. El trabajo estudia cómo se comportan las personas en su interacción con los dispositivos, principalmente dispositivos móviles, con el objetivo de determinar si dicha interacción es producida por una persona física, o si por el contrario se trata de un programa automatizado o aplicación programada (bot).

Para el estudio de la interacción humana con los dispositivos se han analizado bases de datos ya disponibles, concretamente la base de datos HuMIdb (*Human Mobile Interaction database*) [1], que cuenta con registros de interacción de usuarios realizando diferentes tareas en dispositivos móviles, y que será descrita posteriormente. Además, se ha desarrollado un sistema propio de captura de datos en plataformas web, que permite la captura de interacciones producidas desde distintas plataformas.

Para distinguir estas interacciones de las producidas artificialmente, se han desarrollado métodos generativos que permiten imitar la interacción humano-máquina simulando trayectorias o movimientos mediante inteligencia artificial. Estas trayectorias o “ataques simulados” se emplearán junto con los datos “reales” para entrenar modelos de inteligencia artificial y desarrollar técnicas de extracción de características, clasificación y detección de bots.

El desarrollo de los modelos se basa en el uso de redes neuronales adversarias generativas o GAN (*Generative Adversarial Networks*) y en clasificadores realizados con modelos de redes neuronales secuenciales.

Una vez desarrollados métodos de generación de ataques, detección de bots, y la plataforma de captura de datos, se ha desarrollado una aplicación web a modo de demostrador, que permite fusionar el trabajo desarrollado y demostrar la aplicabilidad del mismo. Como prueba de concepto, la web captura datos del usuario, los muestra

por pantalla y realiza predicciones sobre los modelos de redes neuronales previamente entrenados que permiten determinar si la interacción es legítima o automatizada, mostrando el resultado al usuario.

1.1.- Motivación y objetivos del proyecto

El presente trabajo fin de máster pretende servir de continuación o complemento a la investigación realizada por el laboratorio BiDA Lab de la escuela “BeCAPTCHA: Bot Detection based on Behavioral Biometrics” [2] [3]. En dicho trabajo de investigación se propuso la realización de un sistema CAPTCHA invisible al usuario a través del modelado del comportamiento humano en la interacción con la tecnología. Fruto de esta investigación se dispone de la base de datos HuMIdb (Human Mobile Interaction database), descrita en secciones posteriores, la cual se utilizará para parte de la experimentación de este TFM.

Los sistemas CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*) son uno de los métodos más populares para distinguir entre humanos y bots en la red. Son algoritmos que determinan cuándo un usuario es humano, presentándole una serie de desafíos que están asociados a características cognitivas de los seres humanos. Ejemplos de estos desafíos son el reconocimiento de caracteres distorsionados en una imagen o identificar objetos en una imagen. [2]

En el presente proyecto se continúa el desarrollo del proyecto de investigación mencionado, con el objetivo de lograr nuevos métodos de detección de bots, así como métodos de generación de ataques, necesarios para la propia detección. Concretamente, el desarrollo se centrará en el estudio de la interacción táctil (uso de pantallas táctiles de dispositivos móviles) y el sensor de aceleración lineal presente en la mayoría de dispositivos móviles.

Se pretende determinar si existe correlación entre el movimiento de desbloqueo de los móviles, basado en deslizar el dedo hacia la derecha sobre un cuadro en la pantalla, y las trayectorias de sensores inerciales obtenidas cuando se realiza dicho movimiento. Para ello, se comenzará por estudiar cada sensor de forma independiente (pantalla

táctil y acelerómetro), incluyendo la generación de secuencias y detección de bots de forma individual.

Posteriormente se estudiará la posible correlación entre secuencias de ambos sensores registradas en el mismo periodo temporal, y se tratará de desarrollar un método que permita detectar bots en base al estudio de esta información de forma conjunta, en base a la hipótesis de que existe una correlación entre las secuencias registradas. Este método se basará tanto en detección como en generación de secuencias de forma conjunta.

Dado que el desarrollo de estos sistemas requiere de la disponibilidad de información tanto para el aprendizaje como para evaluación, también se tiene como objetivo el desarrollo de una plataforma de captura de datos que registre la interacción de las personas con los dispositivos. El objetivo es realizar un desarrollo web, para hacer la captura de datos compatible con múltiples dispositivos y plataformas. Adicionalmente, se realizará un demostrador sencillo en web que permita mostrar la aplicabilidad del proyecto y realizar una futura campaña de adquisición de nuevos datos.

1.2.- Estado del arte

Actualmente, aproximadamente un 24 % del tráfico web está constituido por sistemas automatizados, también llamados bots, cuyos fines son maliciosos. Algunas de las funcionalidades de estos son la distribución de malware o spam, la vulneración de formularios web, el llenado de bases de datos o la realización de ataques de denegación de servicio distribuidos (DDoS).

Existen mecanismos para la prevención de dichos ataques. Uno de los más conocidos es el llamado CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*: test de Turing público y automático para distinguir a los ordenadores de los humanos). Es un tipo de medida de seguridad que se basa en una autenticación pregunta-respuesta. En él, se le solicita al usuario que complete una prueba sencilla que demuestre que efectivamente es un ser humano y no un robot quien la realiza.

En los últimos años los avances en el campo de la inteligencia artificial han permitido desarrollar sistemas automatizados capaces de burlar este mecanismo de seguridad, obligando a complicar cada vez más las tareas que se solicitan a los humanos. Una alternativa a este sistema es la realización de un *captcha* transparente al usuario, que sea capaz de determinar si la interacción es automatizada o generada por una persona sin someter al usuario a ninguna prueba, solamente a partir del comportamiento observado en la interacción con las tecnologías. Un ejemplo de ello es reCAPTCHA v3, una tecnología desarrollada por Google desde 2018, que devuelve una puntuación con la probabilidad de que la interacción sea producida por un humano.

El TFM propuesto se desarrolla en este contexto, y pretende servir como herramienta complementaria a este tipo de tecnologías de autenticación que resulten “transparentes” al usuario.

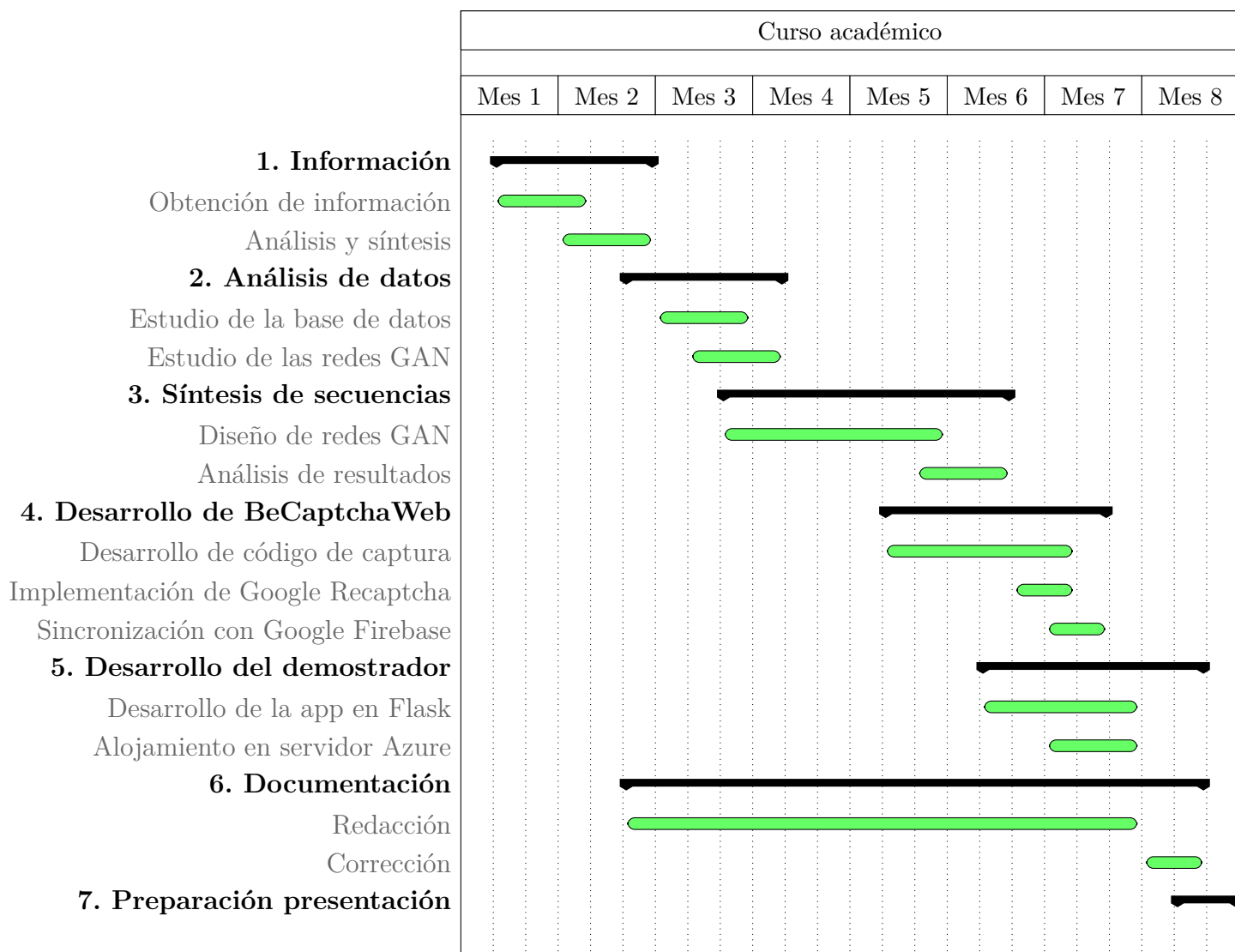
1.3.- Metodología y plan de trabajo

A continuación se describe el proceso seguido para la realización del proyecto durante el curso académico. Para ilustrar las diferentes tareas llevadas a cabo, se muestra a continuación un diagrama de Gantt, donde puede observarse el tiempo de dedicación y la cronología de cada una de ellas.

Como paso previo, es necesario buscar y recopilar información de diferentes fuentes sobre el tema a tratar, así como analizarla y estudiarla. En base a esta información, se escogerán las herramientas, materiales y técnicas más adecuadas a las aplicaciones del proyecto. Una vez escogidas las tecnologías o métodos a emplear, es necesario aprender a utilizarlos o aplicarlos al proyecto. Durante este proceso se ha realizado también la correspondiente documentación.

En primer lugar, se analizó la base de datos disponible, para posteriormente desarrollar modelos de redes GAN, tomando decisiones en base al análisis realizado. Posteriormente se desarrolla la plataforma web de captura de datos, y por último, el demostrador, que permite integrar todo lo desarrollado en una misma aplicación.

El proceso de redacción de la documentación se ha realizado añadiendo contenidos durante el transcurso de los meses. Por último, se ha releído el proyecto para su revisión y corrección, y se ha elaborado y preparado la presentación.



2. Estudio de señales derivadas de la interacción Humano-Máquina con dispositivos móviles

Como se indica previamente, el presente proyecto parte de la investigación realizada por el laboratorio BiDa Lab de la escuela “BeCAPTCHA: Bot Detection based on Behavioral Biometrics” [2] [3]. Durante dicha investigación se recopilaban bases de datos generados a partir de dispositivos móviles, HuMIdb (Human Mobile Interaction database), así como una base de datos que comprende más de diez mil trayectorias de ratón sintéticas y reales, *BeCAPTCHA Mouse database*.

Para la realización de este proyecto se ha empleado únicamente la base de datos basada en la interacción humana con dispositivos móviles, HuMIdb.

2.1.- Descripción de la base de datos

HuMIdb [1] se trata de una base de datos que comprende 14 sensores adquiridos de la interacción de 600 usuarios con sus dispositivos móviles. La información se adquirió a través de una aplicación desarrollada para Android, que pedía al usuario realizar ocho tareas simples con sus propios dispositivos y sin supervisión alguna, una vez al día durante cinco días. La Figura 2.1 muestra una descripción de los sensores capturados en la base de datos.

A continuación se describen las tareas que debía realizar el usuario, durante la captura de los datos. La Figura 2.2 muestra una captura de pantalla de cada una de las tareas en la aplicación.

- a) Se le pide al usuario rellenar datos para capturar tecleo basado en texto fijo (respuesta a datos conocidos) y libre (se le pide que escriba una frase).

Sensors	Sampling Rate	Features	Power Consumption
Accelerometer	200 Hz	x, y, z	Low
L.Accelerometer	200 Hz	x, y, z	Low
Gyroscope	200 Hz	x, y, z	Low
Magnetometer	200 Hz	x, y, z	Low
Orientation	NA	l or p	Low
Proximity	NA	cm	Low
Gravity	NA	m/s^2	Low
Light	NA	lux	Low
TouchScreen	E	x, y, p	Medium
Keystroke	E	key, p	Medium
GPS	NA	Lat., Lon., Alt., Bearing, Accuracy	Medium
		SSID, Level, Info, Channel, Frequency	High
		SSID, MAC	Medium
Bluetooth	NA	Audio	High
Microphone	8 KHz		

Figura 2.1.- Descripción de todos los sensores disponibles en la base de datos HuMIdb.

E = Adquisición basada en evento. Fuente: [1]

- b) y d) Se realizan gestos de interacción táctil hacia arriba y abajo, que consisten en deslizar el dedo por la pantalla para mover el contenido.
- c) Se recopilan pulsaciones de botones en la pantalla
- e) y f) Se le pide al usuario que dibuje un círculo o una cruz en el aire, moviendo el móvil.
- g) Se graba al usuario diciendo la frase "No soy un robot"
- h) El usuario dibuja los dígitos del 0 al 9 en la pantalla, con el dedo.
- Adicionalmente se capturan movimientos de interacción táctil hacia la derecha al final de cada tarea.

Adicionalmente se capturan marcas de tiempo para cada muestra capturada, de cada uno de los sensores. Así como la resolución de pantalla y su tamaño, datos que serán útiles para la realización del proyecto.

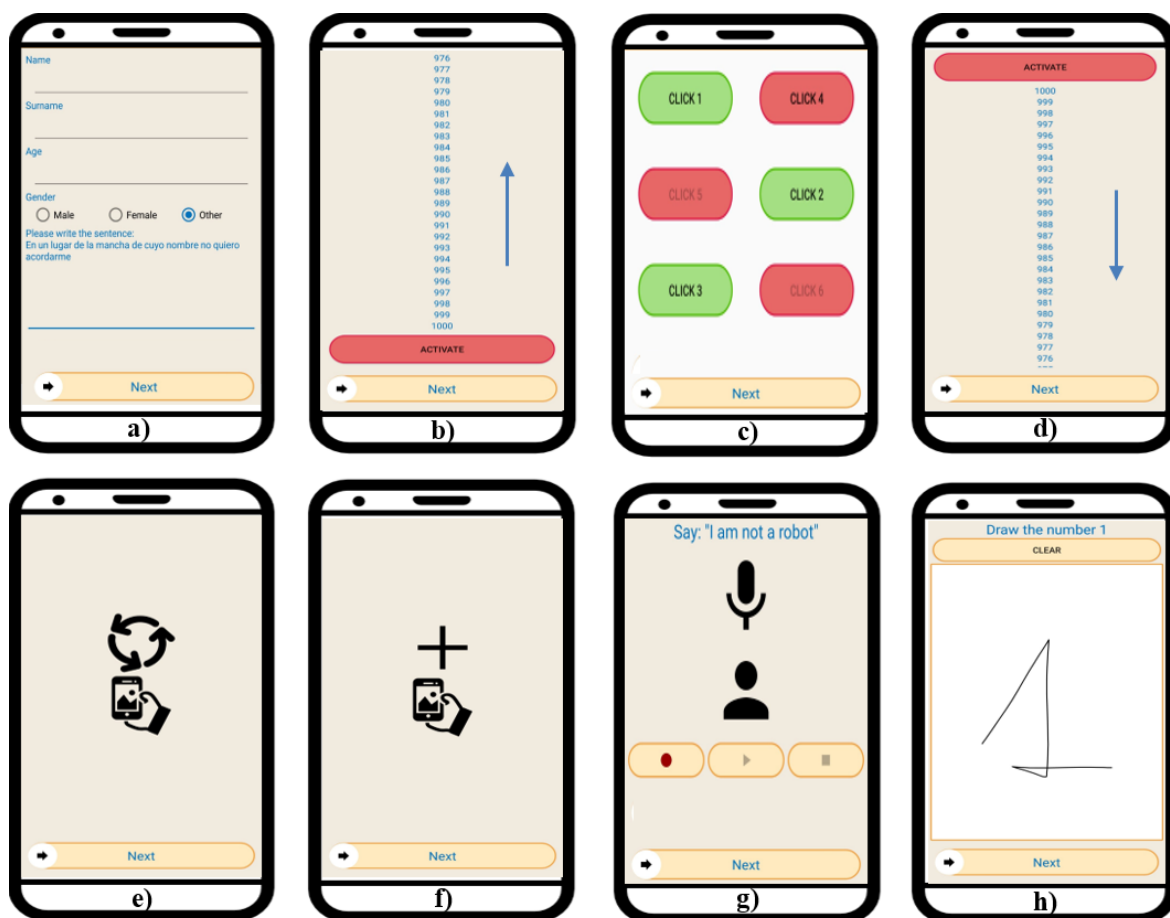


Figura 2.2.- Interfaz de las ocho tareas de la aplicación de captura de los datos de HuMIdb. Fuente: [1]

La estructura de la base de datos se muestra en la Figura 2.3. Existen datos de 600 usuarios, cada uno de los cuales registrará entre una y cinco sesiones. Dentro de cada sesión se registraron en archivos CSV por separado, cada uno de los sensores, dentro de la carpeta asociada a cada una de las tareas llevadas a cabo.

2.2.- Análisis de los datos de sensores

Una vez conocido el contenido de la base de datos, se han seleccionado los sensores que podrían ser útiles para determinar si un usuario es o no un bot, estudiando su interacción con el dispositivo. Dado que se han registrado movimientos de interacción táctil hacia la derecha tras cada una de las tareas, se escoge este “sensor” como punto de partida, para aprovechar el mayor número de datos disponible. Adicionalmente, el acelerómetro se registraba en la base de datos durante todo el tiempo de realización

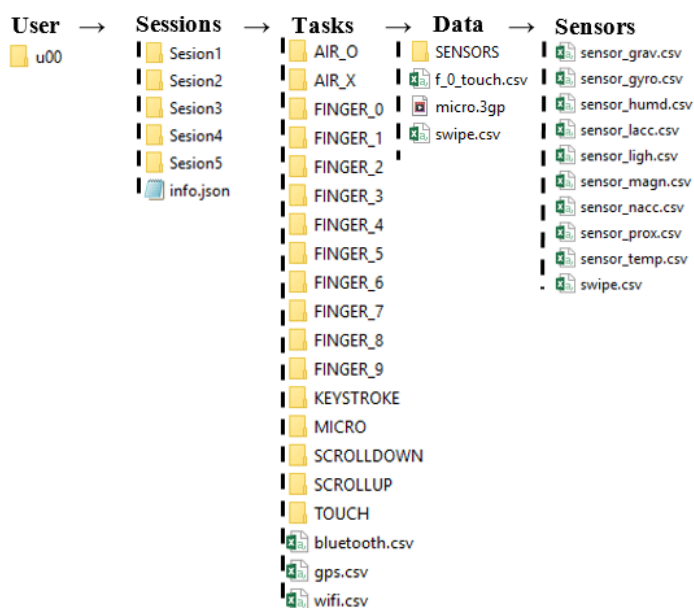


Figura 2.3.- Estructura de carpetas de la base de datos HuMIdb. Fuente: [1]

de tareas por parte del usuario, por tanto, se emplearán también datos de este sensor en el proyecto.

Inicialmente, se ha accedido a los datos de cada uno de los sensores por separado, y se han procesado mediante un script en Python. Posteriormente se tratará de comprobar si existe correlación entre ambos sensores, y en qué medida puede usarse dicha correlación, si existe, para la detección de bots.

2.2.1.- Secuencias de interacción táctil

Las secuencias de interacción táctil capturan las coordenadas X e Y “tocadas” en la pantalla al deslizar el dedo por la misma, así como el “timestamp”, o instante temporal en el cual se captura cada una de ellas.

Los datos se almacenan en los archivos *swipe.csv* dentro del path User → Sessions → Tasks → Data, según muestra la Figura 2.3. Dentro de cada archivo se pueden encontrar los siguientes datos:

- Columna 1: representa el timestamp en milisegundos.
- Columna 2: representa la orientación (vertical = 1, apaisado = 0).

- Columna 3: representa la coordenada x.
- Columna 4: representa la coordenada y.
- Columna 5: representa la presión aplicada.
- Columna 6: representa la acción llevada a cabo (pulsar con el dedo = 0, movimiento = 2, levantar el dedo = 1).

Para la extracción de los datos, es necesario llevar a cabo un procesamiento previo de estos archivos. En primer lugar se separan los distintos movimientos derivados de la interacción táctil que podrían haberse registrado en cada interacción. Esto se hace mediante la columna 6, donde se van cogiendo muestras hasta que el usuario levanta el dedo de la pantalla (columna 6 con valor 1). Es importante destacar que en el caso de este “sensor”, la frecuencia de muestreo no es fija, sino que depende de los movimientos del usuario (basada en evento). Dado que cada usuario dispone de un smartphone propio, las coordenadas X e Y capturadas estarán comprendidas entre 0 y RH para la coordenada X, y entre 0 y RV para Y, siendo RH y RY el número de píxeles en horizontal y vertical de la pantalla (resolución). La columna 2 permite distinguir qué lado de la pantalla es considerado como horizontal o vertical, en base a la orientación del dispositivo. Es necesario normalizar los datos para poder comparar muestras, para lo cual se resta a cada coordenada el primer valor para que la secuencia comience en (0,0) (esquina inferior izquierda de la pantalla) lo cual hace que no influya el punto de partida de la interacción táctil, y se divide cada coordenada entre su resolución (X/RH , Y/RV) para que todas las muestras tomen valores comprendidos entre 0 y 1.

Como paso adicional, se descartaron muestras registradas que contenían únicamente ceros, dado que puede deberse a un error en la captura de los datos. Además, para descartar errores en la captura de los datos, se descartan los movimientos de interacción táctil con un número de muestras inferior a 5.

Como ejemplo, en la Figura 2.4 se muestra la apariencia de algunas muestras de interacción táctil presentes en la base de datos.

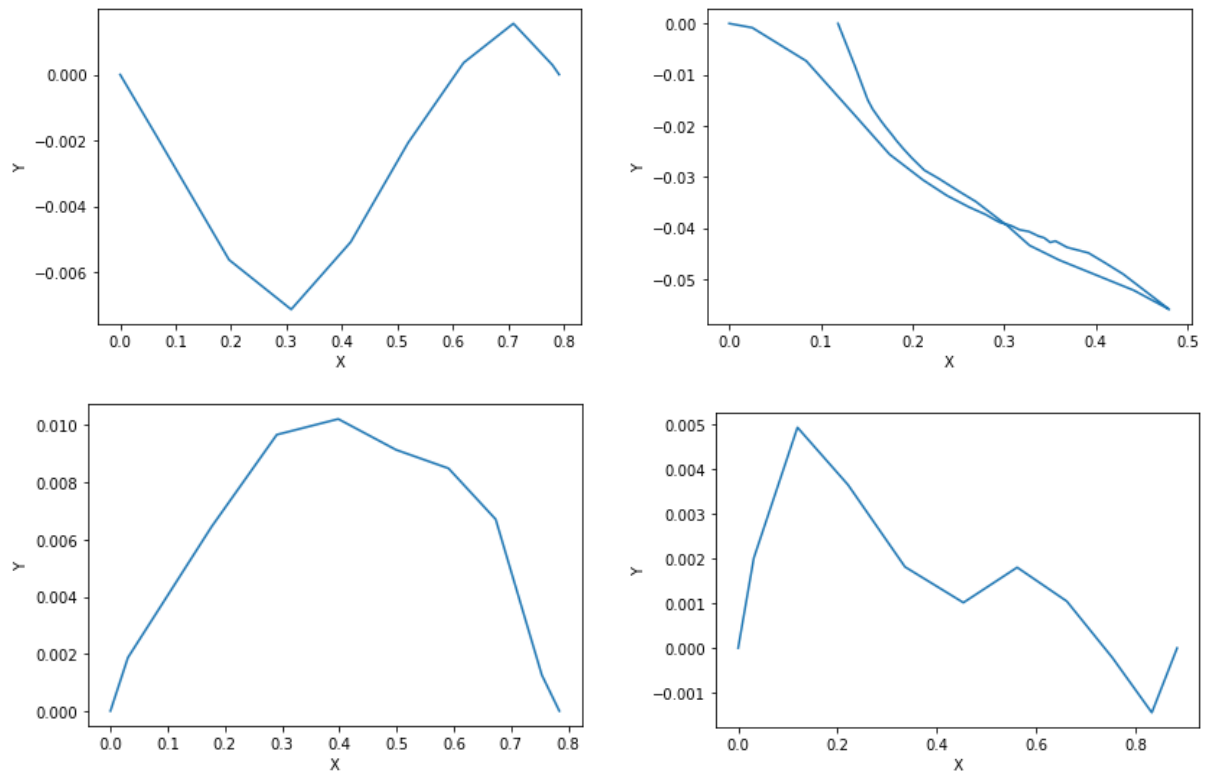


Figura 2.4.- Ejemplos de movimientos de interacción táctil (deslizar el dedo hacia la derecha), disponibles en la base de datos de HuMIdb, tras el procesamiento.

2.2.2.- Secuencias de acelerómetro

Las secuencias de acelerómetro lineal representan la aceleración aplicada al sensor que incluye el dispositivo móvil, excluyendo la fuerza de gravedad, en m/s^2 . Los datos se almacenan en vectores con la aceleración en los ejes X,Y,Z, para cada instante temporal.

En dispositivos móviles Android, el sistema de coordenadas es el empleado en la Figura 2.5, y este no cambia cuando se cambia la orientación de la pantalla.

Los datos se almacenan en los archivos *sensor_lacc.csv* dentro del path User → Sessions → Tasks → Data → Sensors, según muestra la Figura 2.3. Dentro de cada archivo se pueden encontrar los siguientes datos:

- Columna 1: representa el timestamp en milisegundos.

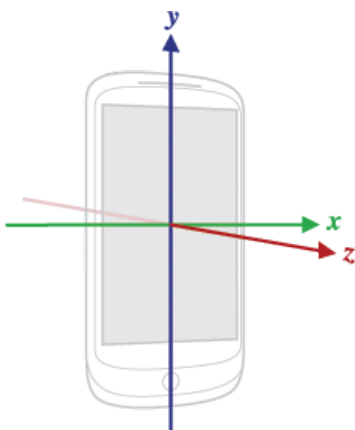


Figura 2.5.- Sistema de coordenadas relativo a un dispositivo móvil, en sistemas Android.

- Columna 2: representa la orientación (vertical = 1, apaisado = 0).
- Columna 3: representa el eje X del acelerómetro lineal.
- Columna 4: representa el eje Y del acelerómetro lineal.
- Columna 5: representa el eje Z del acelerómetro lineal.

Dado que los datos de acelerómetro se capturan durante todas las pruebas realizadas por el usuario en la aplicación, se han empleado únicamente los datos de acelerómetro asociados a un desplazamiento del dedo hacia la derecha (interacción táctil). Por tanto, para cada secuencia de interacción táctil extraída de la base de datos, se extraerán los valores del acelerómetro correspondientes al mismo periodo temporal. Para ello se extraen los timestamps iniciales y finales de cada movimiento de interacción táctil, para después buscar todas las muestras de acelerómetro comprendidas entre esos instantes temporales.

El hecho de capturar los valores de acelerómetro asociados a ese movimiento de interacción táctil permite relacionar los datos capturados con un gesto determinado del usuario.

Destaca que en muchos casos, no se disponía de muestras de acelerómetro para movimientos de interacción táctil, probablemente porque no todos los dispositivos móviles contaban con sensor de aceleración operativo. Por este motivo, el número

de movimientos de acelerómetro disponibles es menor que en el caso de la interacción táctil. Aun así, se dispone de datos suficientes para el proyecto.

Para las secuencias de aceleración lineal, también se aplica una normalización que las hace comenzar en coordenadas ($X=0, Y=0, Z=0$), y que los valores estén comprendidos entre 0 y 1, además de descartarse los movimientos con menos de 10 muestras.

Como ejemplo, en la Figura 2.6 se muestra la apariencia de algunas muestras de aceleración lineal presentes en la base de datos.

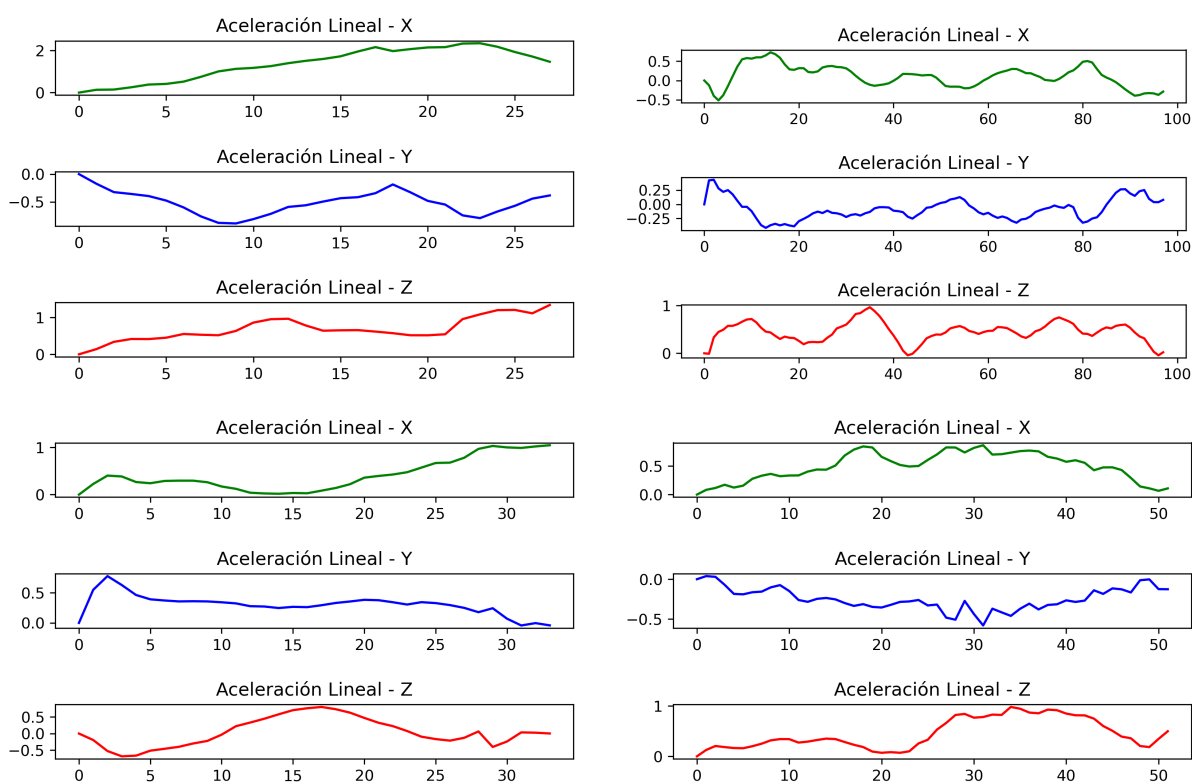


Figura 2.6.- Ejemplos de vectores de aceleración lineal producidos durante una interacción táctil hacia la derecha. Se representa la aceleración normalizada frente al número de muestra en cada instante.

2.2.3.- Análisis estadístico de los datos disponibles de cada sensor

Una vez hecho el pre-procesado de los datos de cada sensor, se han analizado las características de los datos disponibles. Se obtuvieron 35898 movimientos de interacción

táctil hacia la derecha, para cada uno de los cuales se dispone de una serie de muestras de acelerómetro lineal asociadas.

Las muestras de interacción táctil para las cuales no existían datos de aceleración no se han tenido en cuenta para este estudio de la base de datos.

La Tabla 2.1 muestra el numero de muestras totales analizadas de cada sensor, así como la media y desviación típica del número de muestras por cada movimiento. A su vez, se han obtenido el tiempo (T_s) y frecuencia de muestreo (f_s) medio y su desviación típica.

Sensor	Muestras totales	Muestras/movimiento		T_s [ms]		f_s [Hz]	
		Media	D.Típica	Media	D.Típica	Media	D.Típica
Interacción táctil	645981	18.97	4.91	18.23	10.75	61.84	30.11
Acelerómetro lineal	1624471	47.25	22.79	6.59	3.53	193.24	124.03
Ratio Ac./Int. Táctil	2.51	2.49	-	-	-	-	-

Tabla 2.1.- Características de los datos analizados.

Se observa que los movimientos no tienen un número de muestras fijo, y que además, se tiene una varianza elevada, variando el número de muestras de acelerómetro mucho más que en el caso de la interacción táctil. Las Figura 2.7 contiene los histogramas del número de muestras por movimiento para cada sensor.

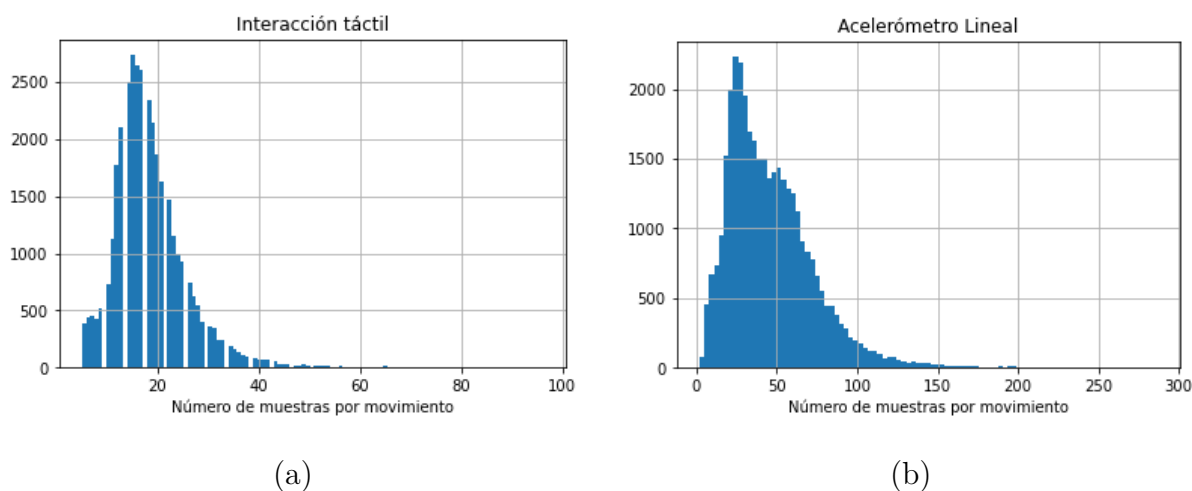


Figura 2.7.- Histogramas del número de muestras por movimiento, (a) Interacción táctil, (b) Acelerómetro Lineal.

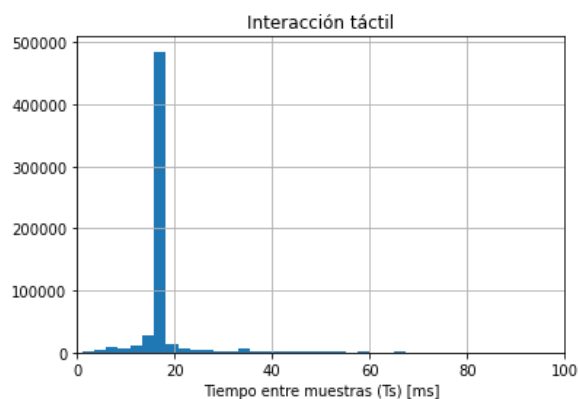
	P10	P90	P25	P75
Interacción táctil	11	27	14	22
Acelerómetro Lineal	18	81	26	61

Tabla 2.2.- Percentiles obtenidos para el numero de muestras por cada movimiento del sensor

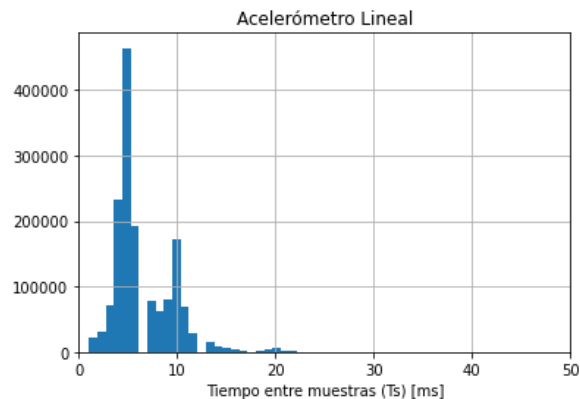
Obtener percentiles de las distribuciones anteriores será de ayuda a la hora de dimensionar parámetros de la red neuronal que se entrenará con posterioridad. Se han calculado los percentiles 10, 90, 25 y 75, obteniéndose los valores de la Tabla 2.2.

Los valores obtenidos indican que el 90 % de los movimientos tiene un número de muestras comprendido entre P10 y P90, y el 75 % de los movimientos tiene un número de muestras comprendido entre P25 y P75.

Analizando la frecuencia de muestreo de los sensores puede verse que esta no es constante. Teóricamente, la frecuencia de muestreo del sensor de acelerómetro en sistemas Android se establece por software en 200 Hz, tal y como se indica en la Figura 2.1. Sin embargo, esta imposición por software viene limitada por las características físicas del sensor que integra cada smartphone. Se ha observado que la frecuencia de muestreo no es constante entre dispositivos, pero tampoco lo es para un mismo dispositivo, existiendo diferencias de pocos milisegundos en el tiempo entre muestras. Esto es debido a que los sensores integrados en los smartphones suelen ser baratos y no muy precisos. Por otro lado, el movimiento de interacción táctil depende de los toques en la pantalla, por lo que no existe una frecuencia de muestreo fija, sino que se registran datos “por evento”. Aun así, en el caso de la interacción táctil el T_s medio es representativo en la mayoría de los casos. Las Figuras 2.9 y 2.8 muestran la distribución de los valores de T_s y f_s para todas las muestras analizadas.

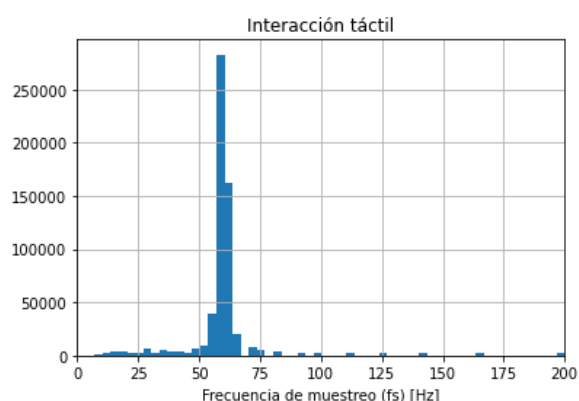


(a)

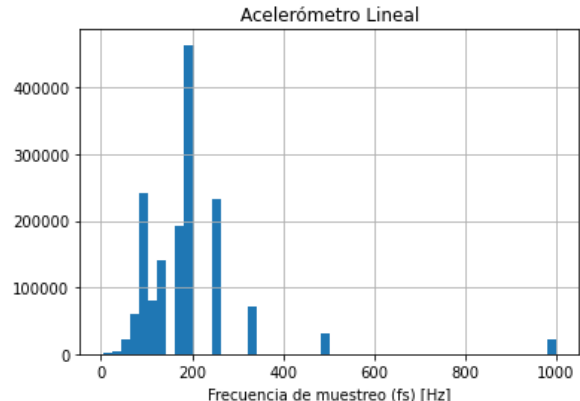


(b)

Figura 2.8.- Histogramas de la distribución del tiempo entre muestras, (a) Interacción táctil, (b) Acelerómetro Lineal.



(a)



(b)

Figura 2.9.- Histogramas de la frecuencia de muestreo, (a) Interacción táctil, (b) Acelerómetro Lineal.

2.2.4.- Extracción de características de los datos: Velocidad instantánea

Dado que las secuencias de interacción táctil y acelerómetro contienen registrada información espacial asociada a instantes temporales, pueden extraerse características como la velocidad instantánea. Esto permite tener un parámetro común para los dos sensores y poder compararlos entre sí.

La velocidad instantánea para cada movimiento en el instante N de un sensor se ha calculado como la distancia euclídea dividida por el tiempo entre muestras (Δ_t) y se calcula para cada muestra (N) con respecto a la anterior ($N - 1$), tal y como muestran las ecuaciones 2.1 para los movimientos de interacción táctil y 2.2 para los de aceleración lineal. Las coordenadas de cada muestra de un movimiento se corresponden con x, y y z .

$$V_{inst_N} = \frac{\sqrt{(x_N - x_{N-1})^2 + (y_N - y_{N-1})^2}}{\Delta_t} \quad (2.1)$$

$$V_{inst_N} = \frac{\sqrt{(x_N - x_{N-1})^2 + (y_N - y_{N-1})^2 + (z_N - z_{N-1})^2}}{\Delta_t} \quad (2.2)$$

La figura 2.10 muestra un ejemplo de la velocidad instantánea calculada para movimientos de interacción táctil y acelerómetro, capturados en el mismo intervalo temporal.

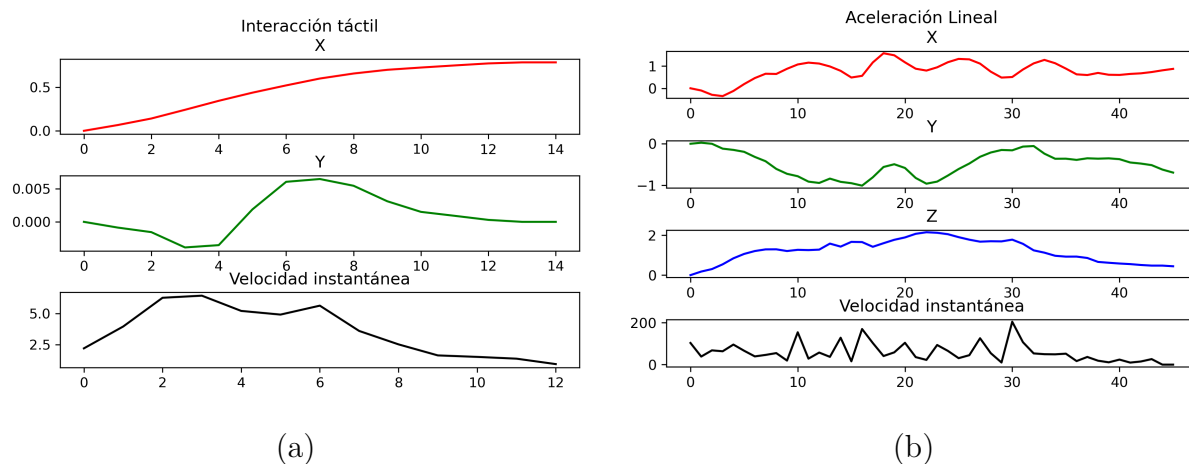


Figura 2.10.- Datos de posición y velocidad instantánea para dos movimientos asociados de interacción táctil y acelerómetro. El eje horizontal representa el número de muestra.

Para el total de los datos, se ha calculado una velocidad media de 2.85 m/s para la interacción táctil y 41.85 m/s para el acelerómetro.

El parámetro de velocidad instantánea será de utilidad para estimar la correlación existente entre los dos sensores.

2.3.- Estudio de la correlación entre secuencias de interacción táctil y acelerómetro

Las secuencias de interacción táctil y acelerómetro se registran en el mismo periodo temporal. La interacción táctil consiste en deslizar el dedo hacia la derecha en la pantalla de un dispositivo móvil. Es habitual que este gesto se haga con el dedo pulgar, moviendo el dispositivo al realizarlo, por lo que se registrarán vectores de aceleración asociados a dicho movimiento. Parece razonable estudiar si existe una correlación significativa entre los datos registrados de estos dos sensores, lo cual podría llevar a tomar decisiones sobre la forma en que se emplearán en aplicaciones de detección de bots. El estudio de la correlación entre sensores se ha hecho mediante parámetros estadísticos, así como empleando redes neuronales.

2.3.1.- Estudio de la correlación mediante parámetros estadísticos

Dado que la interacción táctil viene definida por los vectores X e Y correspondientes a las coordenadas tocadas en la pantalla, y la aceleración se registra en tres vectores X,Y,Z, se ha decidido tomar la velocidad instantánea tal y como se explica en la sección 2.2.4 para el estudio de la correlación entre sensores. De esta forma se tiene una característica común para comparar entre ambos.

Para estudiar la correlación por métodos estadísticos, se han considerado los coeficientes de correlación de Pearson y Spearman, que miden la correlación lineal y no lineal entre dos variables. Ambos coeficientes toman valores comprendidos en el rango $[-1,+1]$, siendo +1 una correlación totalmente positiva, y -1 una correlación totalmente negativa. El valor 0 indicaría que no existe ninguna correlación entre las variables.

El **coeficiente de correlación de Pearson** es una medida de la relación lineal entre dos variables cuantitativas continuas, e indica la intensidad y la dirección de la correlación. La relación entre las variables se considera lineal cuando un cambio en una de ellas provoca un cambio proporcional en la otra variable. El coeficiente, r , se calcula mediante la ecuación 2.3, donde m_x es la media del vector x y m_y es la media del vector y , siendo x e y los vectores de velocidad instantánea de interacción táctil y acelerómetro.

$$r = \frac{\sum (x - m_x)(y - m_y)}{\sqrt{\sum (x - m_x)^2 \sum (y - m_y)^2}} \quad (2.3)$$

El **coeficiente de correlación de Spearman** evalúa la relación monótona entre dos variables continuas. La relación entre las variables se considera monótona cuando estas tienden a cambiar al mismo tiempo, pero a diferente ritmo. El coeficiente, ρ , se calcula mediante la ecuación 2.4, donde D es la diferencia entre estadísticos de orden de x-y, y N es el número de parejas de datos.

$$\rho = 1 - \frac{6 \sum D^2}{N(N^2 - 1)} \quad (2.4)$$

El coeficiente de correlación de Pearson asume que los datos siguen una distribución normal, a diferencia del método de Spearman.

Para observar la correlación entre las secuencias de interacción táctil y acelerómetro es necesario que estas tengan el mismo número de muestras. Dado que la frecuencia de captura del acelerómetro es mucho mayor, se ha optado por submuestrear las secuencias del acelerómetro, de forma que para cada muestra de interacción táctil, se toma únicamente la muestra de acelerómetro más cercana en el tiempo.

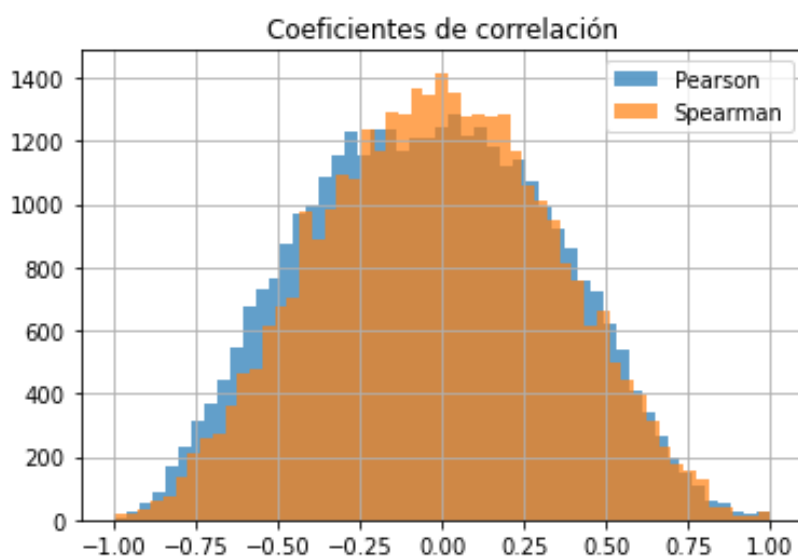


Figura 2.11.- Histograma de los coeficientes de correlación entre las velocidades instantáneas de los sensores de interacción táctil y acelerómetro.

Para cada secuencia de cada sensor, obtenida en el mismo periodo temporal, se han calculado los coeficientes de correlación de Pearson y Spearman. La distribución de los valores obtenidos se muestra en la Figura 2.11. En ella se observa poca o ninguna correlación entre las variables, ya que la mayoría de las secuencias obtienen coeficientes de 0.

Debido al submuestreo realizado en el acelerómetro, no se considera que se pueda tomar este estudio de la correlación como algo concluyente, por lo que se emplearán otros métodos para evaluar la correlación entre las secuencias de los sensores.

2.3.2.- Estudio de la correlación mediante el empleo de aprendizaje supervisado

Para estudiar la correlación mediante técnicas de *machine learning* se emplearán clasificadores binarios basados en modelos de redes neuronales. Para ello, se parte de la base de datos preprocesada de ambos sensores. La idea principal es discernir si existe o no correlación entre secuencias de interacción táctil y acelerómetro, capturadas en el mismo periodo temporal. Para ello se sigue el siguiente procedimiento:

Se divide la base de datos en dos partes. Una de las partes se mantiene sin variaciones, y contiene las muestras de interacción táctil y las asociadas de acelerómetro. En la segunda parte de la base de datos, se cambian las secuencias del acelerómetro por secuencias obtenidas de otro usuario, de forma que ya no se corresponden con las secuencias de interacción táctil. Es decir, los pares interacción táctil-acelerómetro no están capturados en el mismo instante temporal, sino que corresponden a “movimientos” diferentes, capturados quizás por usuarios distintos con diferentes dispositivos. A cada subconjunto se le asigna una etiqueta diferente. Se llamará “original” a la primera parte de la base de datos, e “intercambiado” a la segunda.

Se entrenará un clasificador para que sea capaz de distinguir entre los datos originales y los intercambiados. Si la red neuronal consigue un *accuracy* de más del 50 %, significará que su rendimiento es superior al rendimiento aleatorio, por lo que la red habrá aprendido las diferencias existentes entre los dos subconjuntos. De ser

así, podría considerarse que existe cierta correlación entre secuencias capturadas en el mismo instante temporal. Si el rendimiento es entorno al 50 %, se considerará que no existe correlación alguna, ya que la red no puede distinguir diferencias entre los subconjuntos.

Se han probado tres variaciones de este experimento:

1. División de la base de datos en dos partes o subconjuntos. Se mantiene intacta una parte y en la otra parte se cambian las secuencias del acelerómetro por secuencias obtenidas de otro usuario. Al dividir la base de datos en dos, cada una de las partes tendrá diferentes secuencias de interacción táctil y aceleración.
2. La base de datos se duplica, obteniéndose dos bases de datos iguales. Una de ellas se mantiene sin variación, mientras que se realiza el intercambio de secuencias de aceleración en la otra. Se diferencia del punto anterior en que en ambas clases, los datos de interacción táctil son los mismos y en el mismo orden, y solo se cambian de las secuencias de aceleración de una parte, mientras que en el punto anterior, al dividir en dos la base de datos, los datos de interacción táctil de cada una son distintos.
3. División de la base de datos en dos partes o subconjuntos iguales. No se varía el orden en ninguna de las partes, sin embargo, al dividir la base de datos en dos, cada una de las clases estará entrenada con secuencias de interacción táctil y aceleración que no están en la otra clase. Servirá para contrastar con los datos obtenidos en el punto 1, y ver la influencia de tener datos distintos en cada parte.

El modelo del clasificador empleado es una Red Neuronal Recurrente con la arquitectura que se muestra en la Figura 2.12. El código 2.1 se ha implementado mediante la librería Keras de Python.

La entrada a la red neuronal debe ser un tensor de longitud fija, por lo que se rellenan con ceros las secuencias de acelerómetro e interacción táctil, de forma que todas tengan la misma longitud. Si la longitud de la secuencia es menor, se rellena con ceros hasta ese valor. Si es mayor, se trunca la secuencia. Este proceso se conoce como *zero padding*. Dado que se quiere evitar truncar las secuencias en la medida de

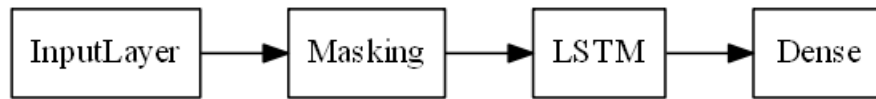


Figura 2.12.- Capas de la red neuronal empleada para el estudio de la correlación entre sensores.

lo posible para evitar la pérdida de información, se ha escogido una longitud de 80 muestras por secuencia. Evaluando los histogramas obtenidos en la Figura 2.7, se sabe que el 100 % de las secuencias de interacción táctil tienen un número de muestras por debajo de 80. En el caso del acelerómetro se trata de un 90 %, con lo que se asegura truncar la menor cantidad de datos, sin añadir excesiva longitud a las secuencias.

Al tener la misma longitud de las secuencias para ambos sensores, se introduce una matriz de entrada de tamaño $[80,5]$, correspondiente a la concatenación por columnas de los valores X,Y de interacción táctil y X,Y,Z de acelerómetro.

Para que la adición de ceros no influya en la clasificación, se ha introducido una capa de *Masking*. Esta capa permite ignorar los ceros añadidos al final de las muestras, de forma que estos no influyan en la actualización de pesos de la red. A efectos de entrenamiento, se considera que las secuencias tienen distinta longitud, ya que los ceros son ignorados. Se utiliza una capa LSTM (*Long Short-Term Memory*) de 64 activaciones, dado que este tipo de capas es eficaz para secuencias temporales y por último una capa densa con una activación sigmoide, que dará como salida un valor comprendido entre 0 y 1.

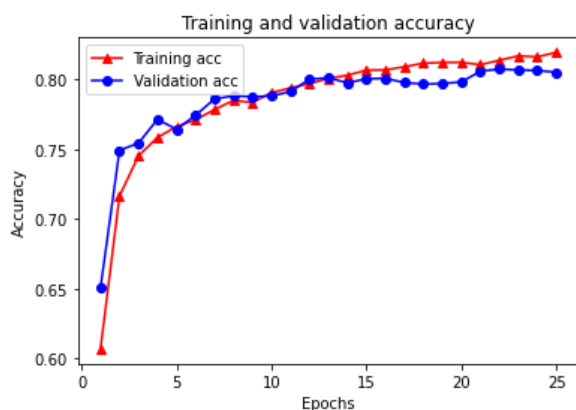
```

Classifier_swipe.input = Input(shape=(80,5),dtype= 'float32')
C = Masking(mask_value=0,input_shape=(80,5))(Classifier_swipe.input)
C = LSTM(units = 64)(C)
C = Dense(1, activation = 'sigmoid')(C)
Classifier_swipe_model = Model(Classifier_swipe.input, C)
Classifier_swipe_model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
  
```

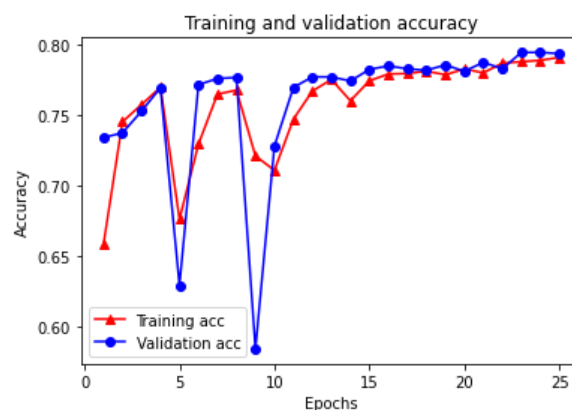
Código 2.1.- Modelo del clasificador empleado para el estudio de la correlación entre sensores.

2.3.2.1.- Resultados obtenidos

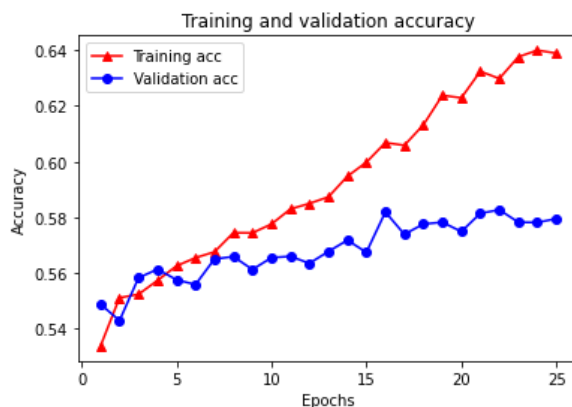
La Figura 2.13 muestra los resultados de entrenar el modelo con las tres variaciones mencionadas anteriormente, entrenando cada caso con batches de tamaño 64 durante 25 épocas.



Caso 1



Caso 2



Caso 3

Figura 2.13.- Resultados de entrenamiento del clasificador para tres variaciones de los datos de entrada.

Se obtienen rendimientos en torno al 80 % de *accuracy* para los casos 1 y 2. El hecho de tomar secuencias diferentes de interacción táctil en cada clase parece influir ligeramente en la decisión de clasificación de la red neuronal, dado que se obtiene un rendimiento de casi el 58 % en los datos de test (Caso 3), lo que supera ligeramente el rendimiento aleatorio. De esta observación se concluye que quizá sea más correcto tomar como referencia el caso 2, para determinar si existe correlación entre las secuencias.

De este análisis se extrae por tanto que la red neuronal sí es capaz de distinguir cuándo las secuencias de acelerómetro fueron capturadas en el mismo periodo temporal que las secuencias de interacción táctil, y por el mismo dispositivo, por lo que no se descarta que exista cierta correlación entre ambos sensores. Sin embargo, cabe la posibilidad de que la red neuronal no esté detectando correlación entre gestos, sino características del dispositivo.

3. Redes Neuronales Generativas (GAN)

Las Redes Neuronales Generativas, *Generative Adversarial Networks* o GANs, son un modelo de aprendizaje profundo (*deep learning*) dentro del campo del aprendizaje automático (*machine learning*).

3.1.- Definición

El modelo de red GAN fue propuesto por primera vez por Ian Goodfellow, en su trabajo *Generative Adversarial Nets*, en 2014. [5]. Es un modelo de tipo generativo, que aprende de los datos de entrada y puede ser usado para generar salidas similares a las aprendidas de la base de datos original.

Dentro de las redes GAN, existen dos submodelos de redes neuronales. El generador, que trata de generar secuencias similares a las reales (procedentes de la base de datos de partida), y el discriminador, que actúa como clasificador entre secuencias reales o sintéticas (creadas por el generador). Los dos modelos se entrenan simultáneamente en un juego de suma cero, donde el generador trataría de engañar al discriminador, generando secuencias cada vez más parecidas a las reales, gracias a la realimentación que le aporta el discriminador a través de su función de pérdidas.

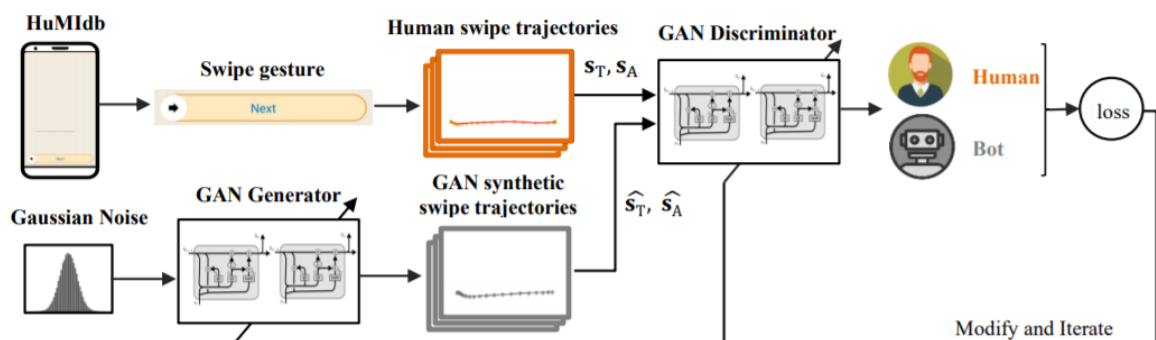


Figura 3.1.- Arquitectura propuesta para entrenar una red GAN. Fuente: [2]

La Figura 3.1 muestra un ejemplo de la estructura de red GAN que se toma como referencia para la realización de este proyecto.

El modelo del generador toma como entrada un vector de longitud fija, y genera una secuencia de la misma longitud. Como vector de entrada se utiliza ruido aleatorio gaussiano. El discriminador recibe como entrada un vector que puede provenir de la base de datos con muestras reales, o bien de los datos generados, y trata de clasificarla como real o sintética. Se trata por tanto de un clasificador binario. Durante el entrenamiento, las pérdidas del discriminador sirven como medida de cuánto se parecen las trayectorias generadas a las reales, y esta información se realimenta al generador, que modifica los pesos de la red para generar trayectorias cada vez más realistas.

El modelo de red GAN engloba al generador y discriminador en un solo modelo, con la relación entre entrada y salida definida en 3.1.

$$Salida = Discriminador(Generador(Entrada)) \quad (3.1)$$

A continuación se describe el proceso seguido durante el entrenamiento de una red GAN, de forma genérica:

- Se genera un batch de tensores de ruido aleatorio gaussiano, con una longitud igual a la de las secuencias reales.
- El generador recibe el batch generado como entrada, y genera una secuencia de salida. En este proceso, el generador no se entrena (no varían sus pesos).
- Se combina el batch generado con uno con secuencias provenientes de la base de datos original (secuencias reales), y se asignan etiquetas a cada tipo de datos. Este vector junto con las etiquetas se introduce como entrada al discriminador, que se entrena en este proceso, comparando las predicciones realizadas con las etiquetas.
- Se genera un nuevo vector con ruido aleatorio gaussiano, al cual se le asocian etiquetas que indican que los datos son reales (de la base de datos original).
- Se entrena la red GAN con ese vector. En este proceso, los pesos del discriminador se congelan, de forma que no se actualizan (no se entrena la red), pero sí se entrena

el generador, en base a la información que proporciona la función de pérdidas del discriminador.

- Este proceso se repite con el siguiente batch, hasta considerar que la red está entrenada.

En el proceso, destaca que al generador nunca se le introducen secuencias reales, ni tiene más información de las mismas que la proporcionada por la función de pérdidas del discriminador. Por otro lado, el batch de entrada al discriminador tiene el doble de tamaño que el de entrada al generador, al combinar tanto secuencias reales como generadas.

El entrenamiento de redes GAN es por lo general complicado, dado que el mínimo de optimización no está fijado, sino que se trata de un sistema dinámico donde el objetivo es conseguir un equilibrio entre dos fuerzas. Por ello, es muy importante realizar un correcto ajuste de los hiperparámetros que configuran las redes neuronales.

3.2.- Hiperparámetros

La información o el aprendizaje de la red neuronal reside en los pesos que se asignan a cada conexión entre las diferentes activaciones de la red. Los hiperparámetros de la red modifican la manera en la que se actualizan dichos pesos durante el entrenamiento. A continuación se describen los hiperparámetros más relevantes a tener en cuenta en el entrenamiento de la red GAN.

Tamaño del batch

El tamaño del batch es el número de vectores de entrada (en el proyecto, el número de secuencias temporales) que se introducen en cada entrada de datos a la red neuronal. El tamaño del batch influye mucho en el entrenamiento de la red, dado que es el número de datos que se consideran en cada actualización de pesos de la red. Tamaños de batch pequeños requieren menor carga computacional, y hacen más rápido el entrenamiento, dado que se traducen en mayor número de cambios en los pesos (para el mismo número

de épocas). Sin embargo, tamaños de batch mayores pueden permitir mejor estimación de los gradientes.

Número de épocas

Se completa una época cuando todos los datos del set de entrenamiento realizan propagación hacia delante y hacia atrás. Una época puede estar compuesta por uno o más “batches”.

Learning rate

Es la tasa de aprendizaje de la red neuronal. Se configura de forma independiente para cada modelo (Generador, Discriminador y GAN). Habitualmente tiene un valor positivo, cercano a cero. Este valor controla la rapidez con la que el modelo se adapta al problema.

Escoger un valor pequeño ralentizará el aprendizaje, haciendo que el cambio de los pesos en *backpropagation* se haga en saltos pequeños (por lo que se corre el riesgo de quedarse en un mínimo local en la función de pérdidas) y por tanto se necesitará entrenar la red durante más épocas. Si se escoge un valor demasiado grande, se obtendrán resultados arbitrarios.

Optimizadores

El optimizador es el algoritmo encargado de realizar la propagación hacia atrás, y cambiar los pesos de la red en base al valor de pérdidas obtenido tras comparar las predicciones con las etiquetas reales.

Tipo, número de capas y número de unidades de cada capa

El tipo de capa dependerá de la naturaleza del problema. Se describirán las capas usadas en la descripción de los modelos propuestos. El número de unidades de cada capa puede considerarse como un filtro, de forma que a mayor éste, mayor información es propagada a las capas sucesivas, y a menor el número, mayor cantidad de información es filtrada. Poner un número de activaciones muy pequeño hace que se pierda información

relevante, mientras que ponerlo muy alto impide a la red centrarse en la información que es realmente relevante para resolver el problema.

Funciones de activación

En cada capa, se realizan transformaciones de los datos de entrada aplicando funciones de activación. La función de activación suele determinar el rango de valores que toma la salida de cada capa, es necesario por tanto que la función empleada se adecue al tipo de problema. La Figura 3.2 muestra tres de las funciones de activación más empleadas.

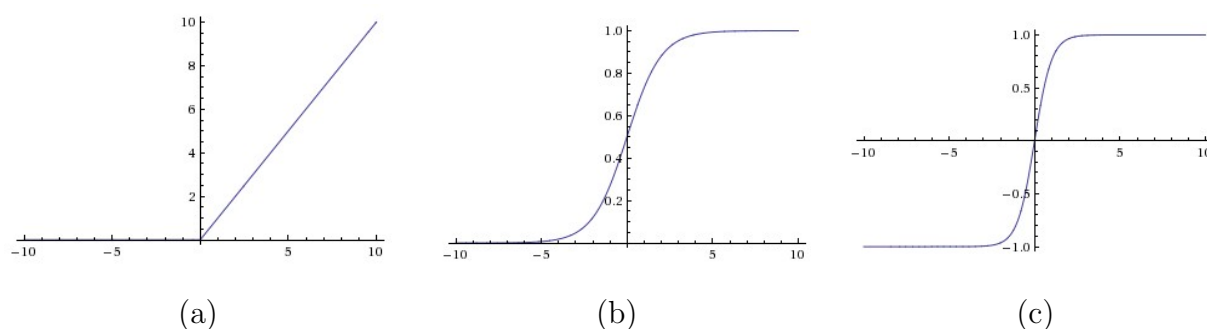


Figura 3.2.- Funciones de activación comúnmente empleadas. (a) ReLu (b) Sigmoide (c) tanh

Funciones de pérdidas

Para saber si el resultado final se parece a lo deseado, se utiliza una función de pérdidas. Esta calcula la distancia entre las predicciones de la red neuronal en base a los datos de entrada, y la etiqueta real asociada a los mismos.

3.3.- Utilidad de las redes GAN en el contexto del proyecto

En el presente proyecto se propone el uso de redes GAN para la mejora de los sistemas de detección de bots. Para ello es preciso partir de información acerca de los posibles ataques que puedan producirse, por ejemplo, en un sitio web o una aplicación móvil. Se entienden como ataques, las interacciones con el dispositivo que han sido

producidas mediante el uso de programas automatizados (bots), simulando interacción humana.

Mediante el entrenamiento de redes GAN es posible generar secuencias de interacción táctil y aceleración que simulen un ataque, a partir del aprendizaje sobre las secuencias reales existentes en la base de datos. Se entrenarán por tanto generadores para la obtención de secuencias sintéticas de ambos sensores.

Al mismo tiempo, la red GAN entrena un discriminador, que permite distinguir entre trayectorias reales y sintéticas. Éste se empleará para realizar la detección, asociando las secuencias generadas con ataques de bots, y las reales con interacciones humanas “legítimas”.

4. Síntesis de secuencias temporales asociadas a la interacción con dispositivos móviles basada en GAN

En este capítulo se describen los modelos de red GAN propuestos. Se ha comenzado desarrollando modelos para la generación y discriminación de secuencias de interacción táctil y acelerómetro de forma aislada, para posteriormente tratar de llegar a un modelo donde se entrenen ambos de forma conjunta, en base al análisis de correlación realizado previamente (sección 2.3).

Durante el desarrollo de los modelos, se emplearán los siguientes tipos de secuencias:

- **Secuencias reales:** Secuencias de interacción táctil y aceleración obtenidas de la base de datos HuMIDB. Permitirán modelar la interacción humana. Se emplearán para entrenar y evaluar el rendimiento de los modelos propuestos.
- **Secuencias sintéticas aleatorias:** Secuencias de ruido aleatorio gaussiano. Se emplearán como entrada a los generadores de las redes GAN, permitiendo el entrenamiento del mismo. También se emplearán para evaluar el rendimiento de los detectores de bots, una vez que éstos han sido entrenados.
- **Secuencias sintéticas generadas por las GAN:** Secuencias generadas por las redes GAN. Permitirán simular un “ataque” o interacción producida por un bot. Se emplearán para entrenar y evaluar el rendimiento de los modelos propuestos.

Se analizará la capacidad de generar secuencias de las redes propuestas, así como la capacidad de distinguir entre estas secuencias y las reales. Además, se propondrán otros modelos de clasificación para la detección de bots.

4.1.- Diseño y entrenamiento de una red GAN para generación y discriminación de secuencias de interacción táctil

Para entrenar la red GAN con secuencias de interacción táctil, se establece la longitud de estas en 30 muestras, en base al análisis de la base de datos realizado. Se realiza el proceso de *zero padding* para rellenar con ceros o truncar las secuencias hasta alcanzar esta longitud. Los datos de entrada tienen dimensiones $[30,2]$, dado que se tienen dos vectores columna con las coordenadas X,Y en la pantalla, de 30 muestras cada uno.

En un primer momento se empleó el procedimiento de entrenamiento descrito en la sección 3.1. Sin embargo, se observaron los siguientes “defectos” tras el entrenamiento:

- La red no aprendía a rellenar las secuencias generadas con ceros (o valores cercanos a cero), con diferentes longitudes, por lo que el numero de muestras de las secuencias generadas era siempre mayor o igual al de las reales, haciendo que el discriminador distinguiese sin esfuerzo entre clases.

Este problema se corrigió modificando la forma en la que se generan los datos desde la entrada de ruido aleatorio hasta la entrada al discriminador (ver Figura 3.1). Para ello, se crea una función que genera batches de secuencias generadas con distinta longitud, a las que posteriormente se le concatenan ceros para lograr que todas las secuencias del batch tengan longitud 30.

- Se hace una predicción introduciendo ruido aleatorio gaussiano por el generador con tamaño de batch 1, lo que genera una secuencia de salida. Se repite un número de veces igual al tamaño del batch. La longitud de esta secuencia de entrada cambia en cada pasada por el generador, y es establecida según una distribución normal $\sim N(18.97,4.91)$, lo cual coincide con la distribución del numero de muestras por movimiento de la interacción táctil observada en la base de datos de HuMIdb. Posteriormente se rellena con ceros cada secuencia hasta obtener longitud 30. De esta manera las muestras generadas se asemejan más a las reales.
- Además, se observó que todas las muestras reales terminaban en una posición con valor $Y=0$, mientras que la red no era capaz de aprender este detalle

(probablemente porque los ceros no son tenidos en cuenta en el entrenamiento). Se obtenían por tanto secuencias rotadas un determinado ángulo con respecto a las secuencias reales.

El problema se corrigió rotando la secuencia generada (justo después de salir del generador y antes de rellenar con ceros) para que la última posición del vector Y fuese cero ($y_N = 0$). Para ello se aplicó una rotación de $(\theta - \alpha)$ radianes a la secuencia, resolviendo la ecuación 4.1, siendo α el ángulo inicial obtenido a partir de la secuencia original según la ecuación 4.2, (x_N, y_N) el último punto de la secuencia original, y (x_0, y_0) el punto inicial.

$$y_{N_{rotado}} = x_N \cdot \text{sen}(\theta - \alpha) + y_N \cdot \text{cos}(\theta - \alpha) = 0 \quad (4.1)$$

$$\alpha = \text{atan}\left(\frac{y_N - y_0}{x_N - x_0}\right) \quad (4.2)$$

Despejando, se obtiene el ángulo θ :

$$\theta = \left[\text{atan}\left(-\frac{y_N}{x_N} + \alpha\right) \right] \text{ (rad)} \quad (4.3)$$

Y se rota la secuencia un ángulo de $(\theta - \alpha)$ radianes, obteniéndose la secuencia rotada definida por los vectores X_{rotado} e Y_{rotado} :

$$\begin{aligned} X_{rotado} &= X \cdot \text{cos}(\theta - \alpha) - Y \cdot \text{sen}(\theta - \alpha) \\ Y_{rotado} &= X \cdot \text{sin}(\theta - \alpha) + Y \cdot \text{sen}(\theta - \alpha) \end{aligned} \quad (4.4)$$

- Destaca que estos cambios se realizan fuera del modelo de red neuronal del generador. No se aplican durante la generación, sino justo después, sobre la secuencia de salida que devuelve el generador.

Los modelos de red generativa, discriminativa y red GAN se muestran en los códigos 4.1, 4.2 y 4.3 respectivamente, donde se muestran los hiperparámetros empleados en cada modelo.

```
Generator_input = Input(shape=(None,2), dtype= 'float32')
G = LSTM(units = 64, activation = 'relu')(Generator_input)
G = Lambda(repeat_vector, output_shape=(None, 2)) ([G, Generator_input])
```

```
G = LSTM(units = 16, activation = 'relu', return_sequences = True)(G)
G = TimeDistributed(Dense(2))(G)
G = Lambda(lambda x: x-x[:,0:1,:])(G)
Generator_model = Model(Generator_input, G, name='Generator')
Generator_optimizer = keras.optimizers.Nadam(lr=0.0002, beta_1=0.5, epsilon=1e-8)
Generator_model.compile(optimizer=Generator_optimizer, loss='mse')
```

Código 4.1.- Modelo generador para el entrenamiento de secuencias de interacción táctil.

En el modelo del generador se emplean dos capas Lambda, que permiten realizar cualquier tipo de operación con los tensores de la red neuronal mediante el uso de funciones de Python. En la primera capa Lambda se realiza una modificación a capa RepeatVector de Keras, que permite emplear batches con secuencias de longitud variable. Esta hace coincidir la salida de la capa anterior con las dimensiones de la siguiente capa, repitiendo el valor. La segunda capa Lambda normaliza la secuencia generada de la misma manera en que se normalizaron las secuencias reales de HuMIdB en el pre-procesado. De esta forma, las secuencias generadas comenzarán en (0,0).

```
Discriminator_input = Input(shape = (30, 2), dtype= 'float32')
D = Masking(mask_value=0,input_shape=(30, 2))(Discriminator_input)
D = Bidirectional(LSTM(units = 64, recurrent_dropout = 0.2))(D)
D = Dense(32)(D)
D = LeakyReLU(alpha = 0.2)(D)
D = Dense(16)(D)
D = LeakyReLU(alpha = 0.2)(D)
D = Dense(1, activation = 'sigmoid')(D)
Discriminator_model = Model(Discriminator_input, D, name='Discriminator')
Discriminator_optimizer = keras.optimizers.Nadam(lr=0.0002, beta_1=0.5)
Discriminator_model.compile(loss= 'binary_crossentropy', optimizer=Discriminator_optimizer)
```

Código 4.2.- Modelo discriminador para el entrenamiento de secuencias de interacción táctil.

Para evitar que la adición de ceros (necesaria para igualar la longitud de las secuencias) influya en el entrenamiento de la red, es necesario añadir como capa de entrada al discriminador una capa de *Masking*. De esta forma se ignoran las etiquetas asociadas a los datos con el *mask_value* indicado (en este caso el 0) durante el entrenamiento. Destaca que la última capa densa cuenta con una única activación sigmoide, que restringirá los valores de salida al intervalo $[0,1]$, y se empleará la entropía cruzada como función de pérdidas, para obtener un clasificador binario.

```
Discriminator_model.trainable = False
gan_input = Input(shape=(30,2))
gan_output = Discriminator_model(Generator_model(gan_input))
gan = Model(gan_input, gan_output, name='GAN')
gan_optimizer = keras.optimizers.Nadam(lr=0.0002, beta_1=0.5, beta_2=0.999, epsilon=1e-8,
    decay=1e-8)
gan.compile(optimizer=gan_optimizer, loss='binary_crossentropy')
```

Código 4.3.- Modelo GAN para el entrenamiento de secuencias de interacción táctil.

El modelo de GAN realiza la operación descrita en la ecuación 3.1, que resulta únicamente en el entrenamiento del generador. Destaca que se establece el discriminador como no entrenable, para que los pesos de este no varíen, ya que este se entrena de forma externa.

4.1.1.- Resultados obtenidos

La red se entrenó durante 450 épocas, con un tamaño de batch de 64. La Figura 4.1 muestra las curvas de pérdidas de los modelos GAN (generador) y discriminador, con el resultado de las pérdidas de cada modelo en cada época, empleando como función de pérdidas *binary-crossentropy* en ambos casos.

Es necesario observar el proceso de entrenamiento para determinar cuándo se debe parar y considerar la red como entrenada. Para ello, durante el entrenamiento, se han ido guardando los pesos de cada modelo de red neuronal, así como las pérdidas obtenidas, y se ha generado una secuencia sintética como referencia, para comparar con las muestras reales y determinar visualmente si la red está obteniendo secuencias

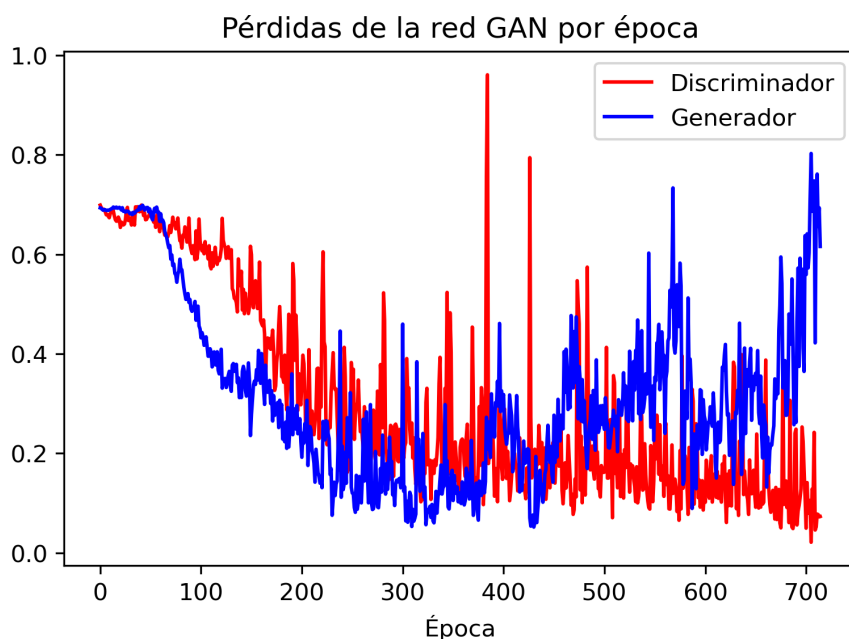


Figura 4.1.- Pérdidas de la red GAN por época.

cada vez más realistas o por el contrario se aleja de los resultados deseados. Cuando se considera que las secuencias son razonablemente parecidas a las reales, y observando las funciones de pérdidas, se para el entrenamiento y se recupera el estado de la red en épocas anteriores si fuese necesario.

Aunque la red se entrenó durante 715 épocas, se ha considerado oportuno recuperar los datos de entrenamiento tras la época 450. Se observó que si se entrenaba más tiempo, las secuencias generadas comenzaban a ser demasiado similares entre sí, de forma que la red tendía a generar casi siempre las mismas secuencias (sobreajuste). Esto, aunque disminuye las pérdidas del discriminador, dado que le es más fácil reconocer las secuencias generadas, es indeseable, ya que se desea que la red sea capaz de generar secuencias con mayor variabilidad. La Figura 4.2 muestra ejemplos de algunas secuencias generadas obtenidas tras el entrenamiento, comparadas con algunas secuencias reales arbitrarias.

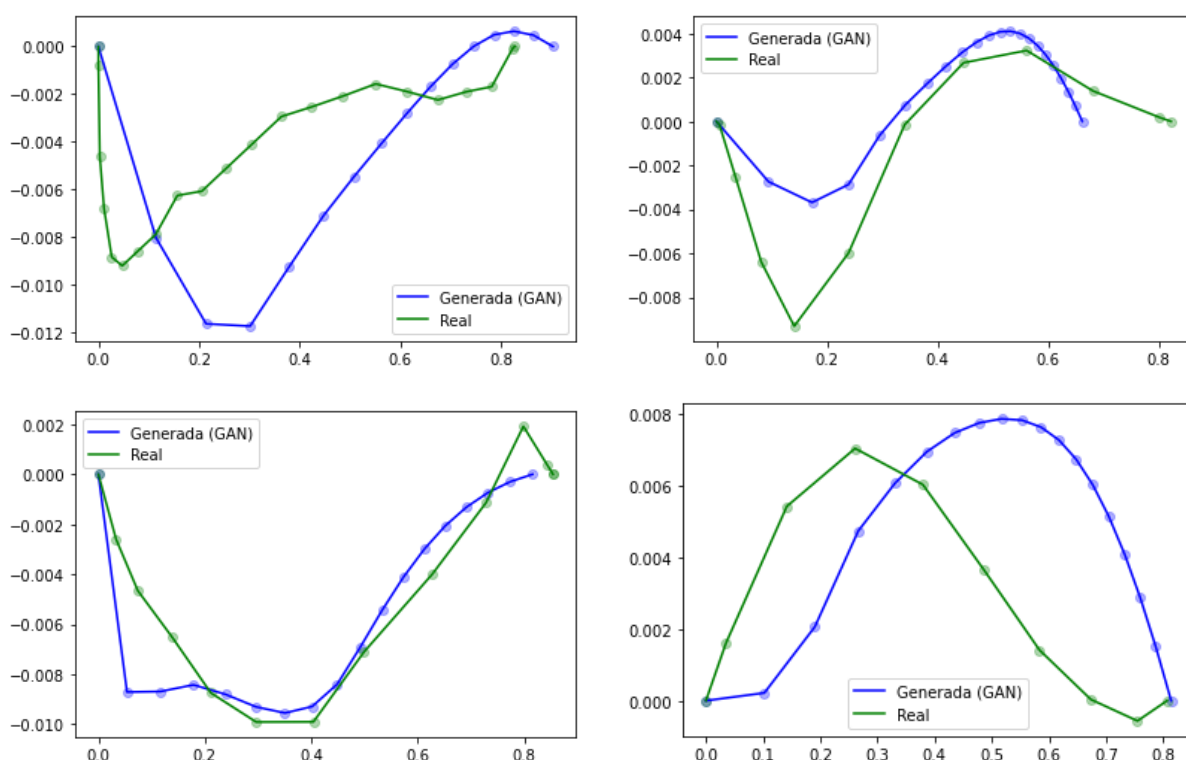


Figura 4.2.- Ejemplos de secuencias de interacción táctil reales frente a las generadas, obtenidas con el modelo descrito.

Además de comprobar visualmente que las secuencias generadas son similares a las deseadas, se han realizado histogramas para observar la distribución de los valores generados frente a la distribución de valores reales, para cada una de las características generadas, en este caso, las coordenadas X e Y de la interacción táctil. Las distribuciones pueden observarse en la Figura 4.3. Para la generación de dicha figura se han generado 25.000 secuencias con la red entrenada, comparándose con el mismo número de secuencias de la base de datos real. Posteriormente se han descartado las muestras añadidas, con valor $(X,Y)=(0,0)$, y se ha tomado el mismo número de muestras para cada distribución, empleando en ambos casos 420.000 muestras.

4.2.- Diseño y entrenamiento de una red GAN para generación y discriminación de secuencias de acelerómetro

En el caso de las secuencias de acelerómetro, se ha tomado una longitud de 80 muestras para el entrenamiento, de forma que los datos de entrada a la red tienen

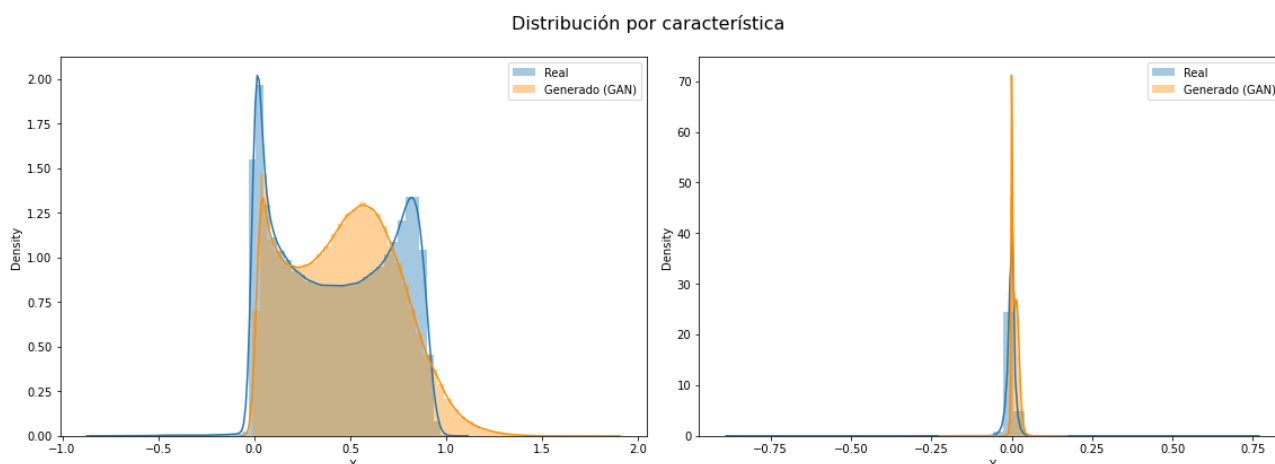


Figura 4.3.- Distribución de los datos reales y generados por característica. Interacción táctil. Izquierda: Valores de X. Derecha: Valores de Y

tamaño [80,3], dado que se tienen tres vectores columna con las coordenadas X,Y y Z de aceleración medida.

En este caso, también se realiza la técnica de *zero padding*, rellenando con ceros las muestras, para lo cual se generan primero secuencias de ruido aleatorio gaussiano con una longitud variable que sigue una distribución normal $\sim N(47.25,22.79)$, similar a la observada en la base de datos de HuMIdb para el sensor de acelerómetro. Estas secuencias se introducen al generador, y posteriormente se añaden ceros a la secuencia de salida del mismo, en un proceso análogo al realizado con las secuencias de interacción táctil.

Para este sensor, los modelos del discriminador y red GAN se han mantenido iguales a los empleados en la interacción táctil, siendo los códigos asociados el 4.2 y 4.3 respectivamente. La única variación es el tamaño de las muestras de entrada, que pasa a ser de (80,3). En el caso del generador, modelo empleado puede observarse en el código 4.4.

```

Generator_input = Input(shape=(None,3), dtype='float32')
G = Bidirectional(LSTM(units = 64, return_sequences = True))(Generator_input)
G = TimeDistributed(Dense(3))(G)
G = Lambda(lambda x: x-x[:,0:1,:])(G)
Generator_model = Model(Generator_input, G, name='Generator')
Generator_optimizer = keras.optimizers.Adam(lr=0.0002, beta_1=0.5)

```

```
Generator_model.compile(optimizer=Generator_optimizer, loss='mse')
```

Código 4.4.- Modelo generador para el entrenamiento de secuencias de acelerómetro.

Este modelo resulta ser más simple, empleándose únicamente una capa LSTM bidireccional y una capa TimeDistributed sobre una capa densa. La capa TimeDistributed permite aplicar la capa densa a cada muestra que recibe como entrada. Por último, se aplica la capa Lambda para normalizar las secuencias, haciéndolas comenzar en el punto (0,0,0).

4.2.1.- Resultados obtenidos

La red se entrenó durante 350 épocas, con un tamaño de batch de 128. La Figura 4.4 muestra las curvas de pérdidas de los modelos GAN (generador) y discriminador, con el resultado de las pérdidas de cada modelo en cada época, empleando también como función de pérdidas *binary-crossentropy* en ambos casos.

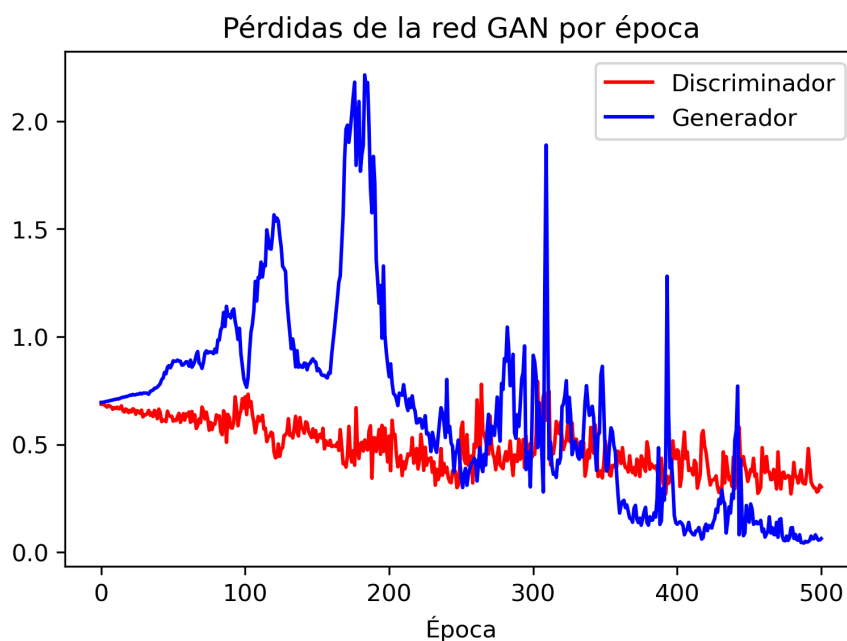


Figura 4.4.- Pérdidas de la red GAN por época.

Observando la Figura 4.4 puede parecer que se mejorarían los resultados entrenando durante más épocas. Sin embargo, la disminución de las pérdidas del generador se debe probablemente a que ha encontrado una manera de burlar al discriminador,

haciendo que este confunda las secuencias generadas con reales. En este punto, comienza a generar secuencias muy parecidas entre sí, dado que resultan efectivas para este propósito, pero afectando drásticamente a la variabilidad de las secuencias. Por este motivo se decide parar el entrenamiento en un punto anterior, donde se tiene mayor equilibrio entre los modelos.

Un ejemplo de secuencia obtenida con el generador tras el entrenamiento se muestra en la Figura 4.5. Debido a la naturaleza ruidosa del sensor de aceleración, resulta complicado determinar visualmente si las secuencias generadas son lo suficientemente similares a las reales, por lo que resulta complicado saber cuándo detener el entrenamiento.

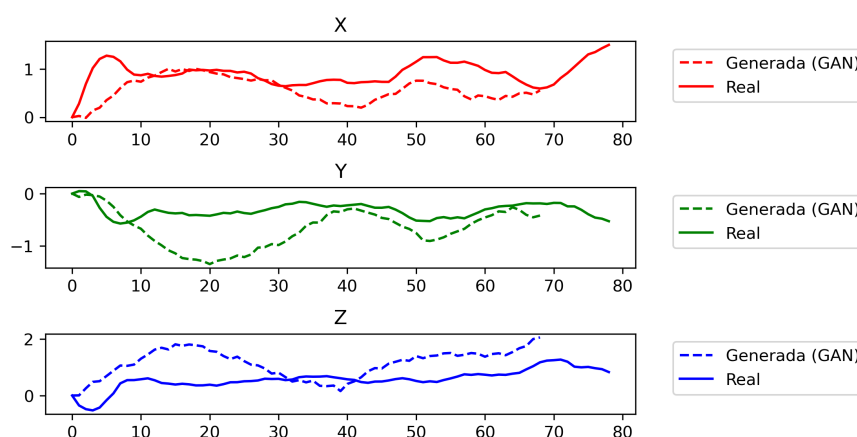


Figura 4.5.- Ejemplo de secuencia de aceleración real frente a una generada, obtenida con el modelo descrito.

La Figura 4.6 muestra la distribución de los valores obtenidos para cada característica, comparando secuencias reales frente a generadas. Para su obtención se consideraron muestras con coordenadas (X,Y,Z) a partir de 25.000 secuencias reales y 25.000 generadas con la GAN. Se eliminaron las muestras añadidas para fijar la longitud de la secuencia, donde $(X,Y,Z) = (0,0,0)$, y se emplea el mismo número de muestras en ambas distribuciones: 1.065.000.

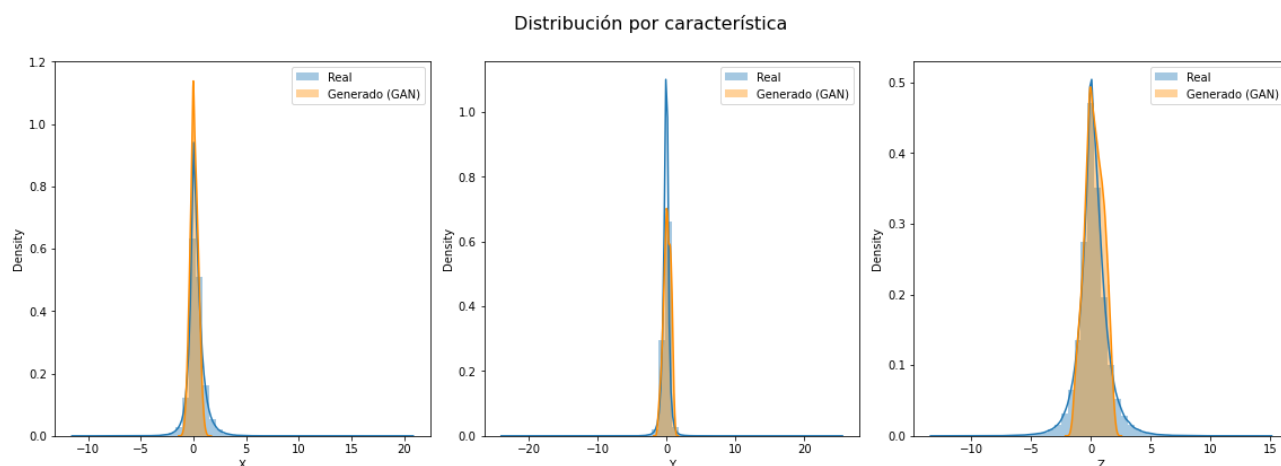


Figura 4.6.- Distribución de los datos reales y generados por característica. Acelerómetro. Izquierda: Valores de X. Centro: Valores de Y, Derecha: Valores de Z.

4.3.- Verificación de las secuencias generadas mediante el empleo de redes neuronales

Para asegurar que las secuencias generadas son correctas se realizó una primera evaluación visual. Sin embargo, se emplearán otros métodos para estudiar más en detalle los resultados obtenidos. Para ello se ha optado por emplear clasificadores basados en redes neuronales.

Se entrenará un clasificador que permita clasificar secuencias reales frente a secuencias de ruido gaussiano. Posteriormente, se introducirán las secuencias generadas mediante la red GAN en el clasificador, obteniéndose una predicción para cada secuencia. El resultado de esta prueba permitirá ver qué tan alejadas están las secuencias generadas del ruido aleatorio, así como el grado de parecido con las muestras reales. Es decir, esta prueba permite evaluar la capacidad de generación de “ataques”, o secuencias sintéticas, y ver hasta qué punto son realistas. La generación de secuencias realistas resulta necesaria para la posterior realización de detectores de bots.

El código empleado para el clasificador se corresponde con el código 4.5, donde NM es el número de muestras de cada sensor (30 para interacción táctil y 80 para acelerómetro) y NC es el número de características, dos para interacción táctil (X,Y) y tres para acelerómetro (X,Y,Z).

```
Classifier_input = Input(shape = (NM,NC), dtype= 'float32')
```

```
C = Masking(mask_value=0,input_shape=(NM,NC))(Classifier_input)
C = LSTM(units = 64)(C)
C = Dense(1, activation = 'sigmoid')(C)
Classifier_model = Model(Classifier_input, C)
Classifier_model.compile(loss= 'binary_crossentropy', optimizer='nadam',metrics=['accuracy'])
```

Código 4.5.- Modelo del clasificador de secuencias reales y ruido gaussiano.

El clasificador se ha entrenado en ambos casos empleando 50.000 secuencias, de las cuales se emplean 10.000 como datos de “test” o verificación y el resto se divide en 26.800 de secuencias de entrenamiento y 13.200 de secuencias de validación.

Las secuencias de validación se emplean para obtener estimar el grado de *accuracy* durante el entrenamiento, por lo que los hiperparámetros de la red se modifican en base a los resultados de predicciones realizadas sobre esos datos. Los datos de validación se emplean una vez ha terminado el entrenamiento, para realizar la clasificación.

4.3.1.- Clasificador de secuencias reales y ruido gaussiano

Para la clasificación, se consideran la siguiente asociación de etiquetas:

- Secuencias reales → Etiqueta “0”
- Secuencias de ruido aleatorio → Etiqueta “1”

4.3.1.1.- Verificación de resultados de interacción táctil

Para las secuencias de interacción táctil, se ha entrenado el clasificador durante 4 épocas, con un tamaño de batch de 64. La figura 4.7 muestra las pérdidas y los resultados de *accuracy* obtenidos. El clasificador consigue un 100 % de *accuracy*, distinguiendo sin problemas entre secuencias reales y ruido aleatorio.

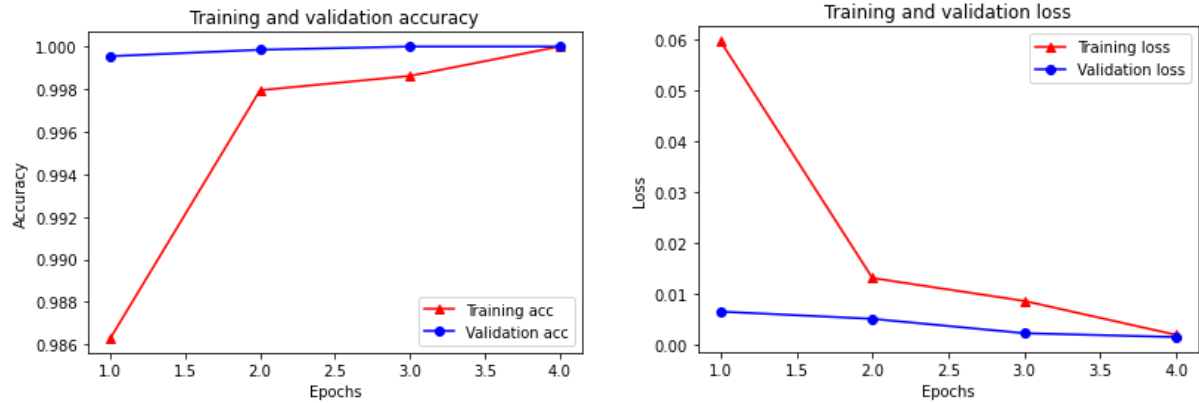


Figura 4.7.- Resultados de *accuracy* y *loss* para el clasificador binario entre secuencias reales y ruido aleatorio gaussiano, entrenado con secuencias de interacción táctil.

A continuación se emplean los datos de test para realizar predicciones sobre el clasificador. Se utilizan 5.000 secuencias de ruido, 5.000 secuencias reales y 5.000 secuencias generadas con la GAN. La Figura 4.8 (a) muestra los valores de etiqueta o *scores* obtenidos para cada secuencia, mientras que la Figura 4.8 (b) contiene los histogramas correspondientes a los *scores* obtenidos para cada tipo de secuencia.

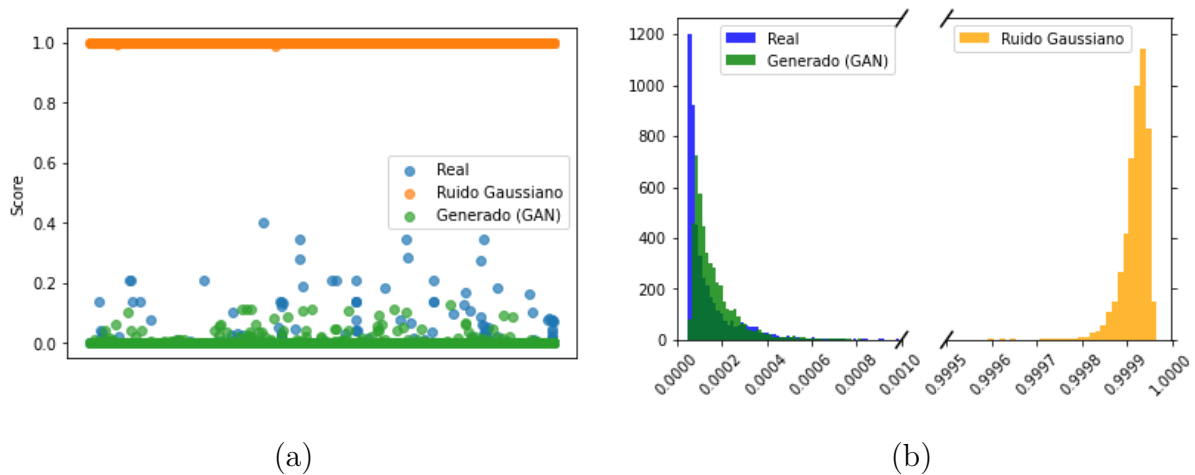


Figura 4.8.- Resultados de clasificación. (a) *Scores* obtenidos para cada secuencia tras la clasificación. (b) Histogramas con los valores de los *scores* obtenidos para cada tipo de secuencia.

En general, los *scores* obtenidos para el ruido son prácticamente 1, mientras que para las secuencias generadas y reales son muy cercanos a cero, con ciertos *outliers*

inferiores a 0.4. A partir de la observación de la Figura 4.8 (a) puede establecerse el umbral de decisión a partir del cual se considerará la secuencia como real o ruido. Si se escoge 0.5 como umbral de decisión, el clasificador habrá clasificado correctamente todas las secuencias de ruido frente a las reales. Además, habrá clasificado las secuencias generadas con la GAN como reales.

Del experimento se extrae la conclusión de que las secuencias generadas a partir de ruido aleatorio distan mucho de este, acercándose mucho a las secuencias reales objetivo.

4.3.1.2.- Verificación de resultados de acelerómetro

Para las secuencias de aceleración, se ha entrenado el clasificador durante 5 épocas, con un tamaño de batch de 128. La figura 4.9 muestra las pérdidas y los resultados de *accuracy* obtenidos. El clasificador consigue un 100 % de *accuracy*, distinguiendo sin problemas entre secuencias reales y ruido aleatorio.

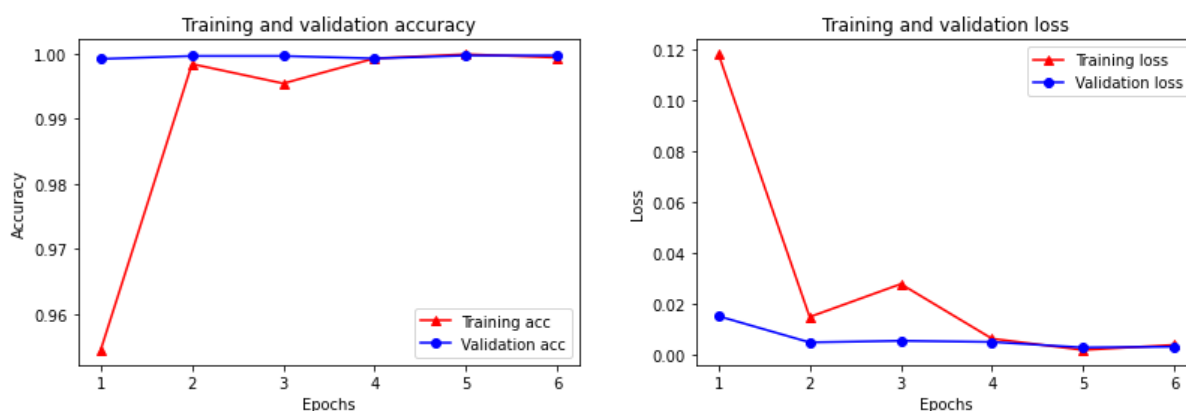


Figura 4.9.- Resultados de *accuracy* y *loss* para el clasificador binario entre secuencias reales y ruido aleatorio gaussiano, entrenado con secuencias de aceleración.

Tal como se hizo para la interacción táctil, se utilizan 5.000 secuencias de ruido, 5.000 secuencias reales y 5.000 secuencias generadas con la GAN. La Figura 4.10 (a) muestra los valores de etiqueta o *scores* obtenidos para cada secuencia, mientras que la Figura 4.10 (b) contiene los histogramas correspondientes a los *scores* obtenidos para cada tipo de secuencia.

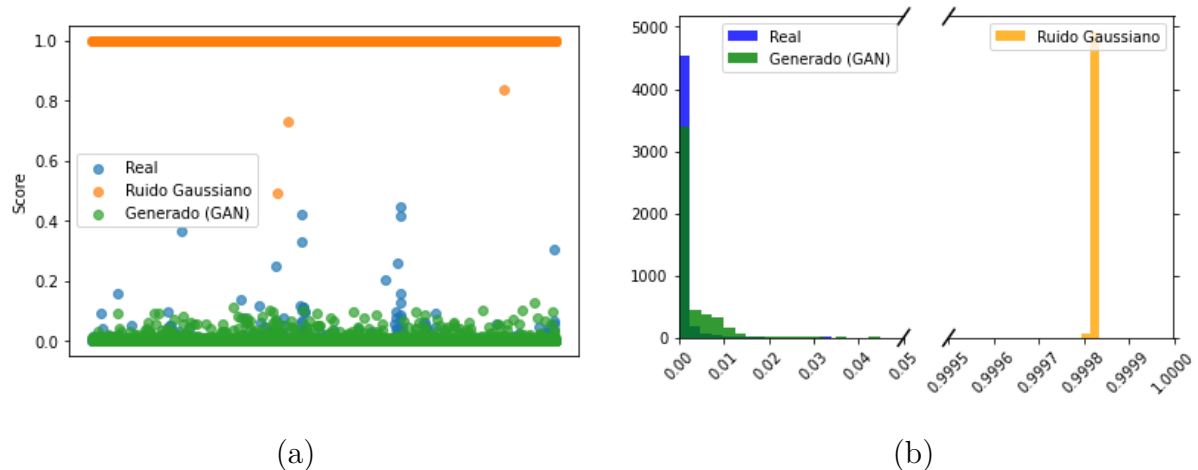


Figura 4.10.- Resultados de clasificación. (a) *Scores* obtenidos para cada secuencia tras la clasificación. (b) Histogramas con los valores de los *scores* obtenidos para cada tipo de secuencia.

En este caso se establece también el umbral en 0.5, para una clasificación correcta entre ruido y secuencias reales. Dado que el acelerómetro es un sensor más ruidoso que la interacción táctil, algunas secuencias reales y de ruido obtienen *scores* cercanos al umbral. En general, las muestras generadas se aproximan mucho más a las reales que al ruido.

4.4.- Verificación de la eficiencia del discriminador en la detección de bots y propuesta de modelos alternativos

En esta sección se probará la eficiencia en la detección de bots de los discriminadores entrenados. Para ello, se simularán dos tipos de ataques mediante el uso de dos tipos de secuencias:

- Bot de ruido: Secuencias de ruido aleatorio gaussiano.
- Bot GAN: Secuencias generadas mediante las redes GAN entrenadas previamente.

Estas secuencias representan las interacciones que se producirían de forma automatizada por los denominados bots. Para probar la detección, se introducirán estas secuencias, junto con secuencias reales a cada discriminador, y se analizarán los resultados obtenidos.

4.4.0.1.- Verificación de resultados de interacción táctil

Se han empleado 5000 secuencias de cada tipo, las cuales se introducen en el discriminador, que predice un *score* por secuencia. El discriminador fue entrenado con las siguientes clases:

- Etiqueta “0”: Secuencias reales \rightarrow Humano
- Etiqueta “1”: Secuencias generadas con la GAN \rightarrow Bot

Los resultados obtenidos se muestran en la Figura 4.11.

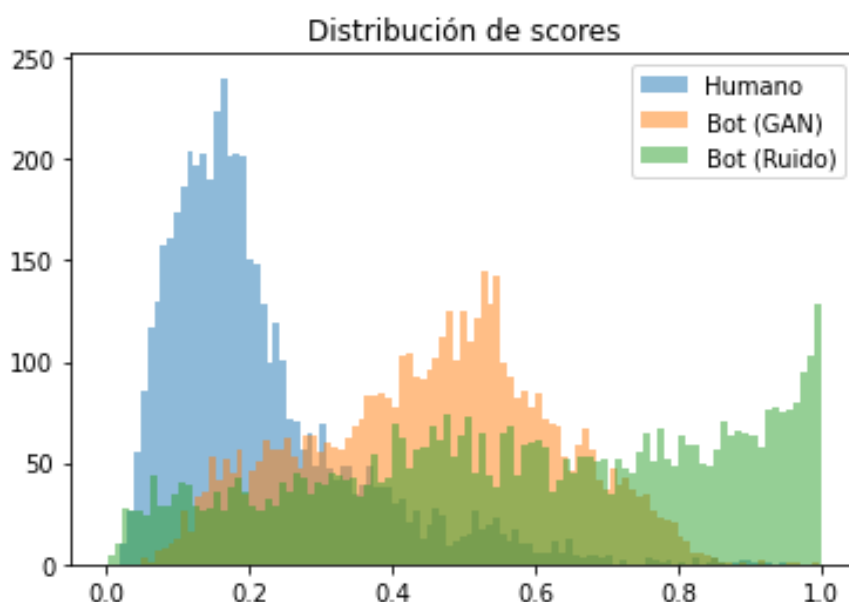


Figura 4.11.- Distribución de *scores* obtenidos tras clasificar 5000 secuencias con el discriminador de secuencias de interacción táctil.

De la observación de resultados, se ha decidido establecer el umbral de clasificación en 0.3, asignando como reales únicamente las secuencias cuyo *score* asociado es menor o igual a 0.3. La Tabla 4.1 muestra la matriz de confusión asociada a dicho umbral de clasificación, tras predecir *scores* para 5000 secuencias de cada tipo.

En general, los resultados de detección son satisfactorios, dado que se pueden detectar ataques con aproximadamente el 80 % de efectividad.

		Predicción		Predicción [%]	
		Humano	Bot	Humano	Bot
Real	Humano	4025	975	80.5 %	19.5 %
	Bot (GAN)	1027	3973	20.54 %	79.46 %
	Bot (Ruido)	950	4050	19 %	81 %

Tabla 4.1.- Matriz de confusión para para el clasificador de secuencias de interacción táctil, con umbral 0.3, para un total de 5000 muestras por clase.

4.4.0.2.- Verificación de resultados de acelerómetro

Para las secuencias de acelerómetro, también se han considerado 5000 secuencias de cada tipo, con la misma asignación de etiquetas, obteniéndose la distribución de scores de la Figura 4.12.

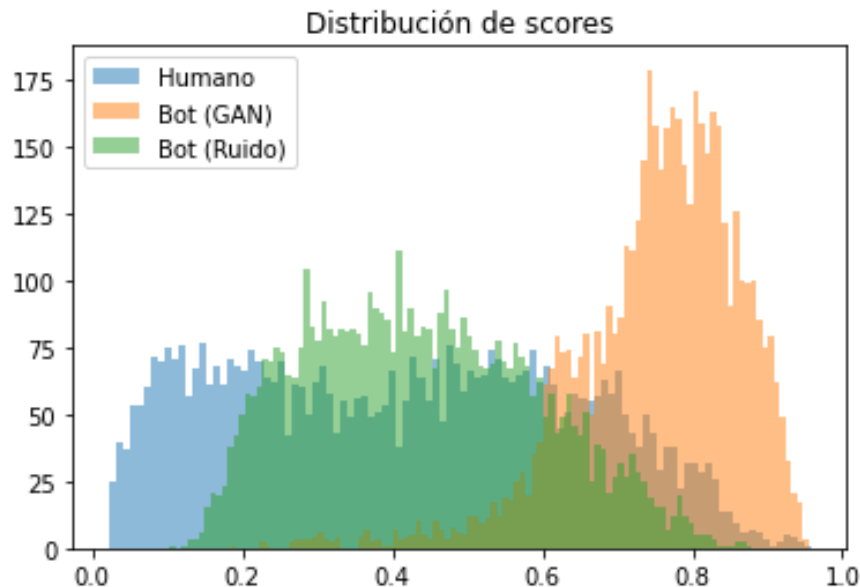


Figura 4.12.- Distribución de *scores* obtenidos tras clasificar 5000 secuencias con el discriminador de secuencias de aceleración.

En este caso se observa que el clasificador confunde habitualmente las interacciones humanas con bots, asignando scores demasiado altos. Sin embargo, reconoce sin problemas las secuencias generadas con la red GAN, clasificándolas como bots. Dado que el clasificador no es capaz de clasificar ruido, los *scores* obtenidos al introducir este

están comprendidos en su mayoría entre 0.2 y 0.8. Esto se debe a que no reconoce el ruido como perteneciente a ninguna de las dos clases con seguridad, por lo que asigna valores intermedios. Si se escoge un umbral de 0.6, se tendría la mejor clasificación entre secuencias reales y generadas por la GAN, sin embargo, la mayoría del ruido sería clasificado como un humano. Si se escoge el umbral 0.3, se estaría dando prioridad a rechazar los ataques de bots, con el coste de confundir más secuencias reales con bots. Las Tablas 4.2 y 4.3 muestran las matrices de confusión obtenidas para los umbrales 0.3 y 0.6 respectivamente.

		Predicción		Predicción [%]	
		Humano	Bot	Humano	Bot
Real	Humano	1831	3169	36.62 %	63.38 %
	Bot (GAN)	21	4979	0.42 %	99.58 %
	Bot (Ruido)	1107	3893	22.14 %	77.86 %

Tabla 4.2.- Matriz de confusión para el clasificador de secuencias de aceleración, con umbral 0.3, para un total de 5000 muestras por clase.

		Predicción		Predicción [%]	
		Humano	Bot	Humano	Bot
Real	Humano	3772	1228	75.44 %	24.56 %
	Bot (GAN)	387	4613	7.74 %	92.26 %
	Bot (Ruido)	4176	824	83.52 %	16.48 %

Tabla 4.3.- Matriz de confusión para el clasificador de secuencias de aceleración, con umbral 0.6, para un total de 5000 muestras por clase.

4.4.1.- Clasificador de secuencias generadas y secuencias reales

Aunque se han obtenido resultados razonables con los discriminadores entrenados, el hecho de entrenar el discriminador junto con el generador empeora la capacidad de detectar de éste. Esto es debido a que para poder generar secuencias lo más realistas posibles, el discriminador debe equivocarse necesariamente. Por ello, se propone

emplear el clasificador del código 4.5 para la discriminación entre secuencias reales y secuencias generadas con la red GAN.

Para la clasificación, se consideran la siguiente asociación de etiquetas:

- Secuencias reales \rightarrow Etiqueta “0”
- Secuencias generadas con la GAN \rightarrow Etiqueta “1”

4.4.1.1.- Verificación de resultados de interacción táctil

La red neuronal se ha entrenado durante 9 épocas, con 5000 secuencias reales y 5000 secuencias generadas con la red GAN. Los resultados de entrenamiento se muestran en la Figura 4.13.

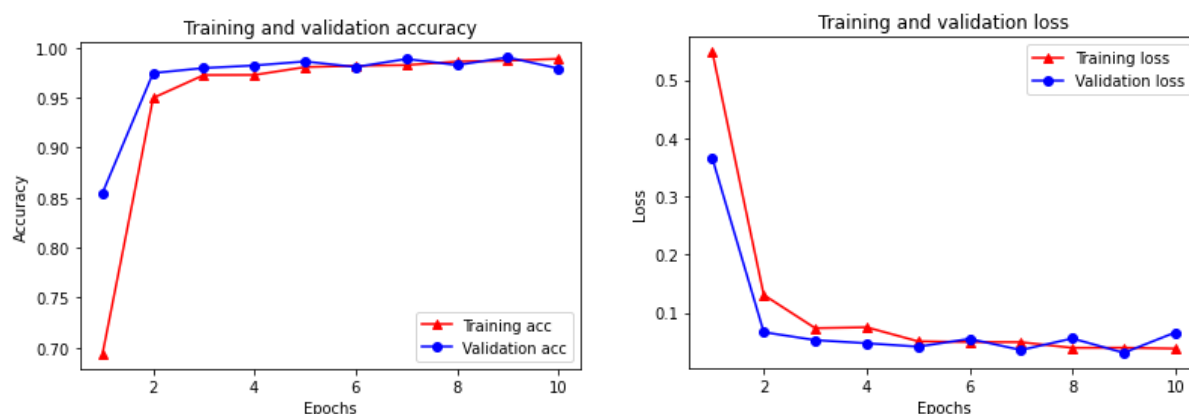


Figura 4.13.- Resultados de *accuracy* y *loss* para el clasificador binario entre secuencias reales y generadas con la GAN, entrenado con secuencias de interacción táctil.

La Figura 4.14 muestra la distribución de scores tras clasificar 5000 muestras de cada tipo tras el entrenamiento del clasificador. De su observación se ha decidido establecer el umbral de decisión en 0.55. La matriz de confusión asociada a este umbral se muestra en la Tabla 4.4.

Se observa que este clasificador mejora la clasificación, aumentando el número de aciertos, respecto al uso del discriminador propuesto en la sección 4.4.0.1. Las secuencias de ruido se clasifican mayoritariamente como producidas por un bot, aunque

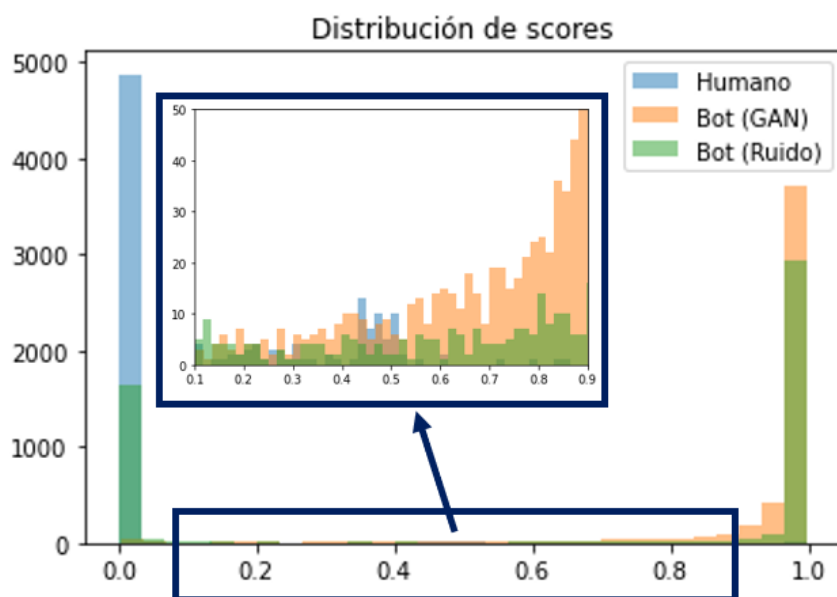


Figura 4.14.- Distribución de *scores* obtenidos tras clasificar 5000 secuencias con el clasificador (Real frente a Generado con GAN) de secuencias de interacción táctil.

el número de secuencias de ruido clasificadas como movimiento humano son elevadas. Sin embargo, la clasificación entre secuencias generadas con GAN y secuencias humanas es bastante precisa, clasificando correctamente más del 95 % de las veces.

Es difícil saber si el éxito en la clasificación se debe a un clasificador bien entrenado, o por el contrario hay demasiada diferencia entre las muestras generadas con la red GAN y las reales.

		Predicción		Predicción [%]	
		Humano	Bot	Humano	Bot
Real	Humano	4977	23	99.54 %	0.46 %
	Bot (GAN)	217	4783	4.34 %	95.66 %
	Bot (Ruido)	1791	3209	35.82 %	64.18 %

Tabla 4.4.- Matriz de confusión para el clasificador de secuencias reales frente a generadas (GAN) de interacción táctil, con umbral 0.55, para un total de 5000 muestras por clase.

4.4.1.2.- Verificación de resultados de aceleración

Se ha repetido el mismo proceso de entrenamiento con el mismo clasificador, esta vez para secuencias de acelerómetro. Se ha entrenado el clasificador durante 14 épocas. Los resultados de entrenamiento se muestran en la Figura 4.15.

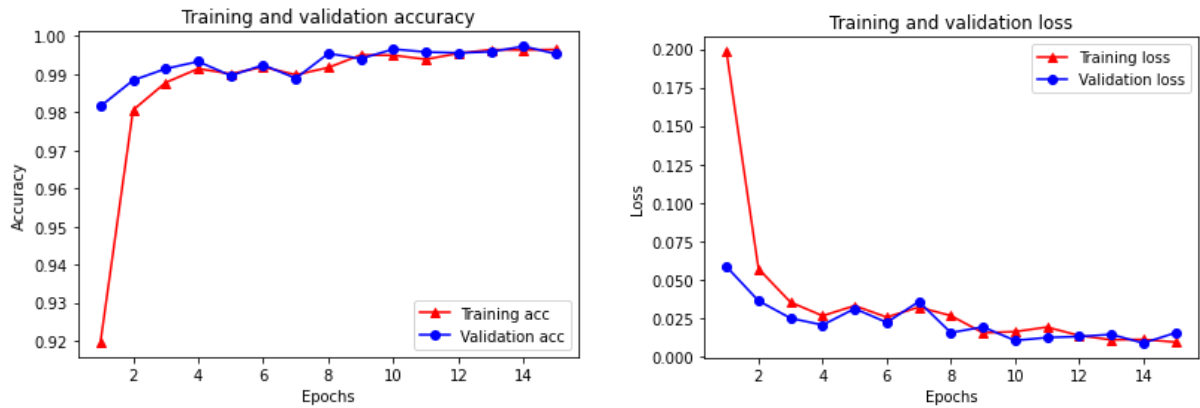


Figura 4.15.- Resultados de *accuracy* y *loss* para el clasificador binario entre secuencias reales y generadas (GAN), entrenado con secuencias de aceleración.

Para la distribución de scores mostrada en la Figura 4.16, se decide escoger un umbral de clasificación de 0.5. La matriz de confusión asociada se muestra en la tabla 4.5.

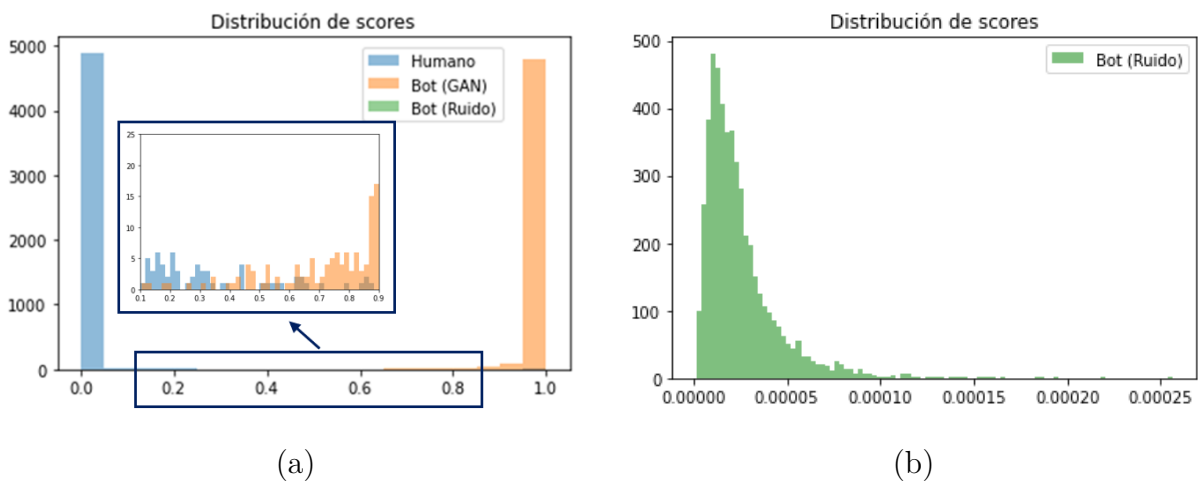


Figura 4.16.- Distribución de *scores* obtenidos tras clasificar 5000 secuencias con el clasificador (Real frente a Generado con GAN) de secuencias de aceleración. (a) Todas las secuencias (b) Zoom de los scores de secuencias de ruido aleatorio gaussiano.

		Predicción		Predicción [%]	
		Humano	Bot	Humano	Bot
Real	Humano	4967	33	99.34 %	0.66 %
	Bot (GAN)	24	4976	0.48 %	99.52 %
	Bot (Ruido)	5000	0	100 %	0 %

Tabla 4.5.- Matriz de confusión para el clasificador de secuencias reales frente a generadas (GAN) de aceleración, con umbral 0.5, para un total de 5000 muestras por clase.

En este caso, aunque los errores en la distinción entre muestras reales y generadas con GAN son mínimos, el clasificador asigna como humanas todas las secuencias de ruido aleatorio gaussiano. Esto puede ser debido a que el detector se ha especializado en detectar las secuencias de la red GAN, sin embargo, no responde correctamente cuando la entrada es ruido aleatorio gaussiano.

4.5.- Diseño y entrenamiento de una red GAN para generación y discriminación de secuencias de interacción táctil y acelerómetro simultáneamente

Tras el estudio realizado en la sección 2.3 no se descartó que existiese correlación entre un movimiento de interacción táctil y los valores de aceleración capturados durante el mismo intervalo temporal. La red neuronal permitía distinguir cuándo estas dos secuencias estaban asociadas en una proporción razonable. De este estudio surge la idea de entrenar una red GAN que incluya información de ambos sensores de manera conjunta.

Se propone el entrenamiento de una red GAN cuya entrada sean matrices de 5 columnas, siendo estas X,Y de la interacción táctil y X,Y,Z de la aceleración. La idea es entrenar la GAN para que sea capaz de distinguir interacciones humanas de bots, pero analizando ambos sensores a la vez, en lugar de hacerlo por separado.

Esta propuesta tiene una serie de desafíos, a diferencia de los modelos de GAN entrenados previamente:

1. Las secuencias de interacción táctil y aceleración tienen distinta longitud. Aunque puede aplicarse *zero padding* como se ha hecho previamente, no será posible añadirlos de forma externa a la red neuronal del generador.

En las redes entrenadas anteriormente, se pasaron batches de ruido de longitud 1 por el generador para posteriormente añadir ceros hasta rellenar un número de muestras fijo, dado que la red neuronal no aprendía a rellenar ceros correctamente. El número de muestras de los batches era variable, en función de la media y desviación típica del número de muestras de cada sensor.

En el entrenamiento conjunto, no será posible realizar esto, dado que no pueden introducirse a la red diferentes longitudes para cada sensor.

2. No puede introducirse el giro de las secuencias de interacción táctil, puesto que se aplicaba de forma externa a la GAN, tras el paso de la secuencia por el generador y antes de la adición de ceros.
3. El entrenamiento de los sensores está basado en modelos de generador distintos. El número y configuración de las capas de los generadores propuestos por separado es distinto. Como solución a este problema, puede diseñarse un modelo de red neuronal funcional, que separe en dos caminos distintos los vectores asociados a cada sensor, para pasar las entradas por distintas capas y posteriormente unir el vector de salida. La diferencia con respecto a implementar dos generadores independientes es que el modelo podría aprender las correlaciones de forma natural.

Teniendo estas limitaciones en cuenta se han hecho diferentes pruebas, entrenando dos modelos distintos de red GAN. En ambos, se ha mantenido el discriminador sin variación, y se ha adaptado el modelo del generador con diferentes configuraciones.

4.5.1.- Modelo de generador secuencial

Los modelos secuenciales se basan en pasar los datos de entrada por capas sucesivas, de forma que la salida de una capa es la entrada de la siguiente.

Se ha probado a entrenar dos modelos secuenciales con las capas que muestra la Figura 4.17.

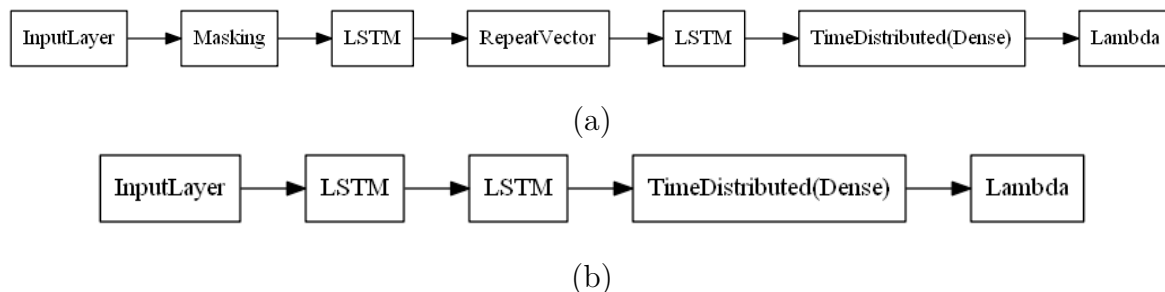


Figura 4.17.- Modelos secuenciales probados para el generador de la GAN propuesta, para el entrenamiento de dos sensores simultáneamente.

Se ha probado a generar secuencias introduciendo diferentes ruidos a la entrada del generador.

1. Vector de ruido aleatorio gaussiano de longitud fija.
2. El ruido de entrada se genera de longitud variable, según la media y varianza de cada sensor. Posteriormente se añaden ceros a la secuencia de entrada al generador para rellenar con una longitud fija, y se concatenan los vectores en una matriz de entrada de cinco columnas. Se añade la primera capa de *Masking* al generador para no tener en cuenta los ceros en el entrenamiento, y se utilizan activaciones ReLu en las capas. Esta opción pretende simular el paso de secuencias de longitud variable por el generador. Las activaciones *ReLu* mantienen a cero los valores nulos, mientras que el *masking* los ignora en el entrenamiento.

Se ha observado que la introducción de la capa *RepeatVector* mejora las secuencias de interacción táctil, empeorando las de aceleración. Si se deja de usar esta capa, se obtiene el efecto contrario. La capa *RepeatVector* “suaviza” las secuencias, eliminando el “ruido” que puedan tener, lo cual es más favorable en la interacción táctil. Dado que el sensor de aceleración es más ruidoso, el uso de esta capa perjudica los resultados.

No se ha logrado encontrar una configuración de capas en el generador que logre un equilibrio en la generación de secuencias de ambos sensores. Por ello se propone un modelo funcional, que tenga en cuenta estas diferencias observadas.

4.5.2.- Modelo de generador funcional

Los modelos funcionales no siguen un orden secuencial a la hora de introducir los datos por las capas de la red. En el modelo propuesto, los datos se dividen en dos caminos, de forma que pueden escogerse distintas capas para entrenar cada sensor, y posteriormente concatenarlas para tener una matriz de cinco columnas como entrada al discriminador.

El esquema propuesto para este modelo del generador se muestra en la Figura 4.18.

En este modelo, los datos de entrada con tamaño $[80,5]$ pasan por capas Lambda que separan los datos en dos caminos, con tamaños $[80,2]$ y $[80,3]$, cada uno de los cuales representa las secuencias X,Y de interacción táctil y X,Y,Z de aceleración. La salida se concatena para volver a obtener secuencias tamaño $[80,5]$. La última capa Lambda permite normalizar las secuencias generadas, de forma que todas empiecen con la primera coordenada en cero.

En la generación de las secuencias de interacción táctil, la capa Lambda entre las dos LSTM realiza la función de la capa *RepeatVector* de Keras, empleando la salida de la LSTM y el tamaño de secuencias de la capa Lambda asociada.

Se ha entrenado la red GAN con este modelo de generador, con diferentes configuraciones de hiperparámetros, y durante distinto número de épocas. Se observó que si se sobreentrena el modelo, el generador comienza a generar secuencias cuyos valores se disparan, o secuencias completamente aleatorias, por lo que es necesario parar el entrenamiento antes de que esto ocurra. Sin embargo, no se ha logrado generar secuencias satisfactorias.

La Figura 4.19 muestra un ejemplo de dos secuencias generadas conjuntamente, para cada sensor, a partir del entrenamiento de la GAN durante 220 épocas, con tamaño de *batch* de 64. Puede observarse que los resultados distan mucho de lo obtenido entrenando cada sensor por separado, no consiguiendo tanto parecido con las secuencias reales. En este modelo se entrena con longitud de secuencias fija, de 80 muestras. Por lo observado, la red no consigue aprender a rellenar con ceros las secuencias a partir

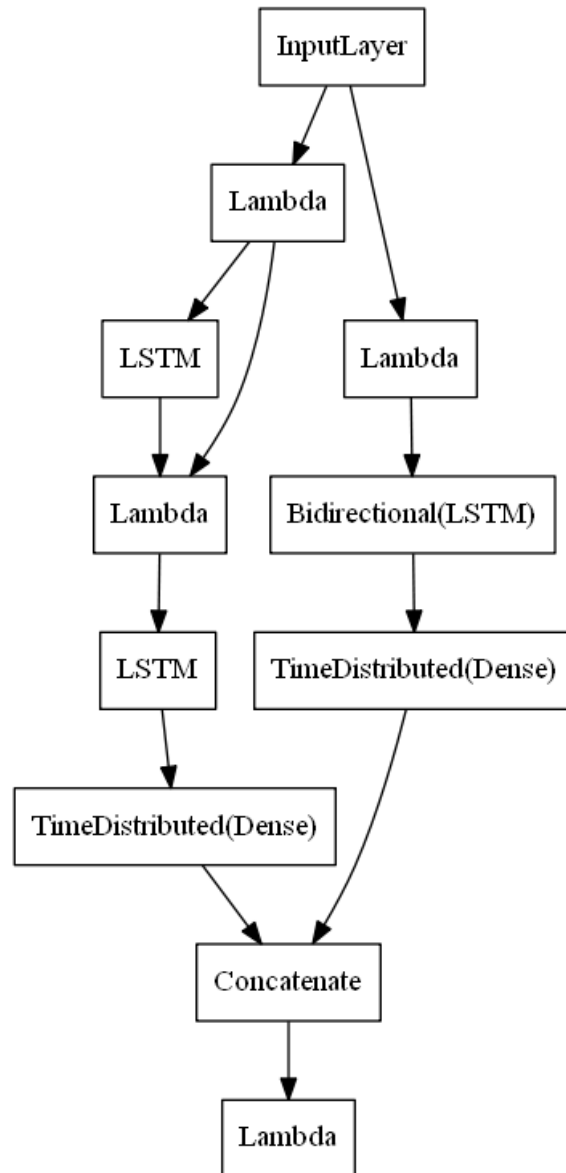


Figura 4.18.- Modelo funcional propuesto para el generador de la red GAN. Camino izquierdo: Entrenamiento de interacción táctil. Camino derecho: Entrenamiento de aceleración.

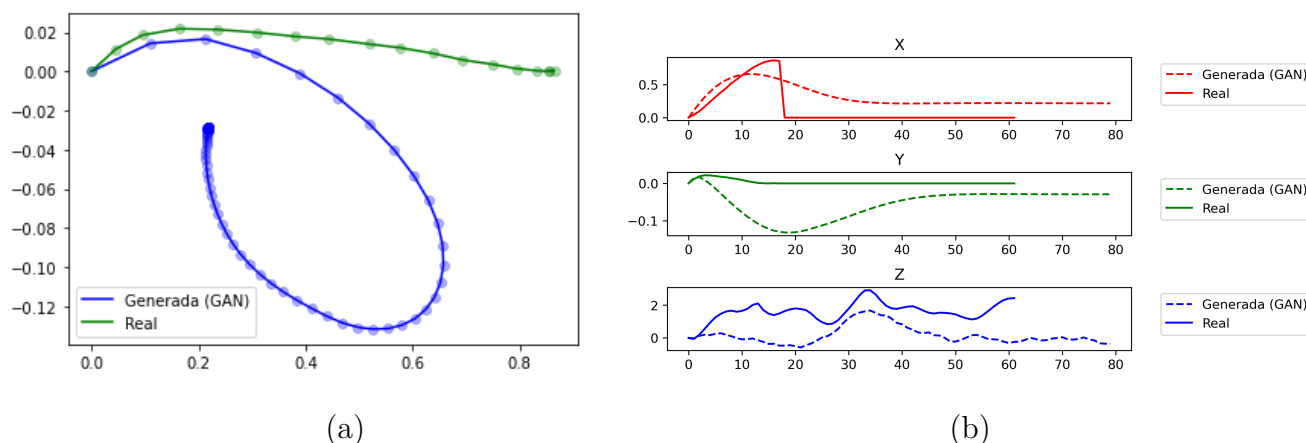


Figura 4.19.- Ejemplo de secuencias obtenidas con la GAN con modelo de generador funcional. (a) Interacción táctil (b) Aceleración

de un cierto valor, aunque si que consigue mantener un valor constante cercano a cero al final de la secuencia en algunos casos.

Dado que visualmente no se obtienen los resultados esperados, no se ha continuado estudiando las secuencias obtenidas. No obstante, no se descarta que pudieran obtenerse resultados válidos si se dedica más tiempo a la configuración de la red neuronal.

4.5.3.- Conclusiones

Tras las diferentes pruebas, no se han obtenido resultados satisfactorios en cuanto a la generación de secuencias. En el uso de un modelo secuencial, se ha visto que la elección de las capas condiciona mucho los resultados finales. El uso de determinadas capas favorece la generación óptima de un sensor, perjudicando al otro. No se ha logrado encontrar el equilibrio que permita generar secuencias válidas en ambos sensores.

En el caso de modelos funcionales, donde se dividen las capas de la red en dos “caminos”, es posible seleccionar las capas y número de activaciones que mejor se adaptan a cada sensor. Sin embargo, el aprendizaje está basado en la evaluación conjunta de ambas secuencias por parte del discriminador. Es decir, el generador se entrena en base a la función de pérdidas del discriminador, que recibe a su entrada las secuencias de ambos sensores al mismo tiempo, por lo que resulta difícil que la actualización de los pesos de toda la red lleve a secuencias óptimas en ambos sensores.

Además, dado que la generación se divide en dos caminos, no queda claro que exista la correlación necesaria entre las secuencias de ambos sensores.

Sería necesario profundizar más en la configuración de los modelos, o el modo en que se entrenan las redes GAN para tratar de mejorar los resultados. Dado que no se pueden generar secuencias similares a las reales, no se considera el estudio del discriminador para la detección de bots, dado que estaría basado en discriminar ataques poco realistas.

5. Desarrollo de la plataforma de captura de datos BeCaptcha Web

Como objetivo en el presente trabajo fin de máster, se ha propuesto el desarrollo de una plataforma web de captura de datos, a partir de los cuales se podrá realizar entrenamiento y evaluación de detectores de bots.

El uso de una página web permite el acceso desde cualquier plataforma o dispositivo, siempre y cuando este cuente con un navegador web. De esta forma se tiene una vía de captura de datos multiplataforma, que permite capturar las interacciones con independencia del sistema operativo (SO), ya sea este Android, iOS, Windows, Linux, Mac OS u otro. Esto representa una ventaja frente al desarrollo de una aplicación específica para un SO, como se trata de la desarrollada para la captura de los datos presentes en HuMidB. La obtención de los datos a partir del navegador web es totalmente transparente al usuario, dado que prácticamente la totalidad de los usuarios de smartphone cuentan con navegadores preinstalados en sus dispositivos y por tanto no se requiere la instalación de software adicional. Además, la detección de bots es comúnmente empleada en páginas web, donde se requiere discernir si el usuario es una persona, o por el contrario se trata de un software automatizado.

No obstante, la captura de datos vía web presenta una desventaja importante, y es que existe un amplio abanico de navegadores web disponibles en el mercado, cada uno de los cuales cuenta con particularidades a la hora de leer los sensores presentes en el dispositivo. En la sección 5.1 se describe el panorama actual en cuanto a la obtención de datos vía web, así como las limitaciones existentes a la hora de realizar esta implementación.

5.1.- Obtención de datos de sensores vía web

Los navegadores web son aplicaciones encargadas de gestionar peticiones web y servir los contenidos solicitados al usuario. Son por tanto *clientes*, que realizan peticiones a un equipo *servidor*, donde está alojada la página web. El servidor envía los archivos asociados a la página web solicitada, y el navegador los interpreta y los muestra por pantalla. Es en la interpretación de dichos contenidos donde existen diferencias entre navegadores.

Para extraer información de la interacción humano-navegador, se emplearán Interfaces de Programación de Aplicaciones (APIs por sus siglas en inglés), estas son construcciones disponibles en los lenguajes de programación que permiten a los desarrolladores crear funcionalidades complejas de una manera simple. Estas abstraen el código más complejo para proveer una sintaxis más fácil de usar en su lugar. [10] Se emplearán APIs integradas en los navegadores web, empleando código Javascript, dado que está incorporado por defecto en los navegadores, de forma que la mayoría de ellos pueden interpretarlo.

La elección de las APIs a emplear se tomará intentando maximizar la compatibilidad entre navegadores, es decir, que se seleccionarán las APIs que sean compatibles con la mayoría de navegadores web para cada sensor. Además, se ha consultado cuáles son los navegadores más empleados en el mundo en la actualidad. La Tabla 5.1 muestra la participación en el mercado (en inglés *market share*) de los navegadores más conocidos. Los datos se obtienen de contabilizar la proporción de usuarios que visitan un grupo de páginas web, empleando un navegador web en particular. En concreto, los datos se han obtenido de estudios llevados a cabo por StatCounter Global Stats [8] y NetMarketShare [9] en marzo de 2021 y noviembre de 2020.

Destaca una cierta “dominación” del mercado por parte del navegador Google Chrome, tanto en dispositivos móviles como en equipos de sobremesa. Dado que la elección de la API web condiciona el funcionamiento de la captura de datos, se elegirá la API que mayor compatibilidad tenga con los navegadores más usados, aunque dando prioridad a Google Chrome como navegador objetivo, por ser el más usado a nivel

Navegador	Todas las plataformas		Navegadores de PC		Navegadores de smartphone	
	StattCounter	NetMarketShare	StattCounter	NetMarketShare	StattCounter	NetMarketShare
	Marzo 2021	Marzo 2021	Marzo 2021	Marzo 2021	Noviembre 2020	Noviembre 2020
Chrome	64.19 %	66.43 %	67.14 %	69.33 %	61.35 %	70.46 %
Safari	19.03 %	12.74 %	10.11 %	3.28 %	25.68 %	13.98 %
Edge	3.74 %	5.07 %	8.47 %	11.56 %	-	-
Firefox	3.68 %	3.62 %	7.95 %	6.30 %	0.5 %	0.31 %
Opera	2.13 %	1.51 %	2.61 %	0.92 %	1.91 %	2.12 %

Tabla 5.1.- Porcentajes de uso de los navegadores web en el mundo, para diferentes plataformas.

global. Esta decisión garantiza el funcionamiento de la plataforma de captura para la mayoría de los usuarios.

5.2.- Obtención de datos procedentes de la interacción Humano-Dispositivo

En la presente sección, se detalla la manera en la que se ha desarrollado la plataforma de captura de datos, así como el tipo de datos capturados. Las funciones principales que proporciona la plataforma desarrollada son:

- Captura de datos de diferentes sensores, provenientes de la interacción con dispositivos, ya sean estos PCs tradicionales o dispositivos móviles.
- Implementación de la herramienta CAPTCHA: Google reCAPTCHA v3. La implementación de esta tecnología permite obtener una puntuación que evalúa si la interacción ha sido producida por un humano o por un bot, en base al análisis del comportamiento en la web que hace Google. Esta tecnología y su implementación se describirán más en detalle en la sección 5.2.5.
- Los datos capturados se enviarán a una base de datos que se actualizará en tiempo real. Para ello ha sido escogida la tecnología de Google Firebase. En concreto, la base de datos de Google Realtime Database. Esta tecnología y su implementación se describirán más en detalle en la sección 5.2.6.

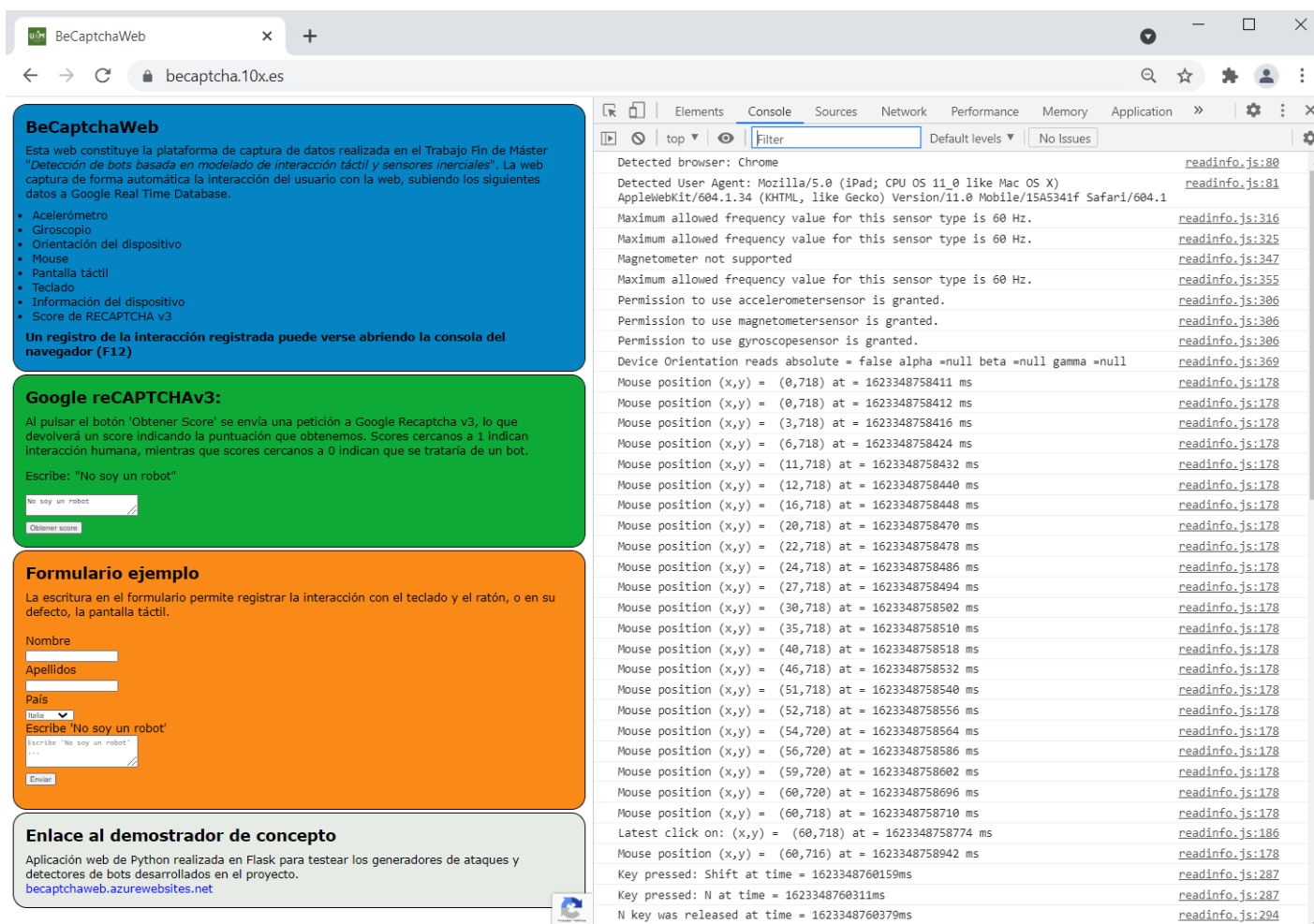
5.2.1.- Tecnologías empleadas en el desarrollo de la web

La página web se ha alojado en un servidor, empleando una Raspberry Pi 3B+ con sistema operativo Raspbian Stretch. Para ello se ha configurado un virtualhost de Apache. Se han abierto los puertos 80 y 443 en el router para proporcionar acceso a la web desde Internet, y se ha empleado freedns.afraid.org como DNS Dinámico, para poder alojar la página web en la dirección <https://becaptcha.10x.es>. Los certificados digitales necesarios para una conexión segura con HTTPs se han generado empleando *certbot*. De esta manera se consigue un acceso a la web desde cualquier dispositivo de Internet.

Dado que el objetivo de la web es la captura de datos, el trabajo está centrado en ese objetivo, y no tanto en la apariencia o contenido de la web. De esta forma, se introducen en la web únicamente algunos elementos como formularios, desplegables y botones, para que el usuario interactúe con ellos, generando así datos que registrar. A modo de depuración, los datos que se registran se van mostrando en la consola de depuración que incluyen los navegadores, y que habitualmente se muestra pulsando la tecla F12. La Figura 5.1 muestra una captura de la apariencia de la página web, así como los datos registrados en la base de datos, vistos desde la consola del navegador Google Chrome. El uso de esta consola ha sido de utilidad para la depuración a la hora de desarrollar la web, mientras que tener la web accesible desde Internet desde el principio ha permitido hacer pruebas con dispositivos móviles reales.

Para el desarrollo de la plataforma se han escrito los siguientes archivos:

- *index.html* y *stylesheet.css* Archivo HTML cargado al entrar en la web, así como la hoja de estilo CSS asociada.
- *readinfo.js* Script que controla la captura de datos, y su subida en tiempo real a Google Firebase.
- *recaptchav3_control.php* Script para realizar consultas a Google reCAPTCHA v3. Se describirá más en detalle en la sección 5.2.5.
- *getDatabase.py* Script de Python que permite descargar la base de datos almacenada en Google Realtime Database.



BeCaptchaWeb

Esta web constituye la plataforma de captura de datos realizada en el Trabajo Fin de Máster "Detección de bots basada en modelado de interacción táctil y sensores inerciales". La web captura de forma automática la interacción del usuario con la web, subiendo los siguientes datos a Google Real Time Database.

- Acelerómetro
- Giroscopio
- Orientación del dispositivo
- Mouse
- Pantalla táctil
- Teclado
- Información del dispositivo
- Score de RECAPTCHA v3

Un registro de la interacción registrada puede verse abriendo la consola del navegador (F12)

Google reCAPTCHA v3:

Al pulsar el botón 'Obtener Score' se envía una petición a Google Recaptcha v3, lo que devolverá un score indicando la puntuación que obtenemos. Scores cercanos a 1 indican interacción humana, mientras que scores cercanos a 0 indican que se trataría de un bot.

Escribe: "No soy un robot"

No soy un robot

Obtener score

Formulario ejemplo

La escritura en el formulario permite registrar la interacción con el teclado y el ratón, o en su defecto, la pantalla táctil.

Nombre

Apellidos

País

Escribe "No soy un robot"

Enviar

Enlace al demostrador de concepto

Aplicación web de Python realizada en Flask para testear los generadores de ataques y detectores de bots desarrollados en el proyecto.

becaptchaweb.azurewebsites.net

Detected browser: Chrome

Detected User Agent: Mozilla/5.0 (iPad; CPU OS 11_0 like Mac OS X) AppleWebKit/604.1.34 (KHTML, like Gecko) Version/11.0 Mobile/15A5341f Safari/604.1

Maximum allowed frequency value for this sensor type is 60 Hz.

Maximum allowed frequency value for this sensor type is 60 Hz.

Magnetometer not supported

Maximum allowed frequency value for this sensor type is 60 Hz.

Permission to use accelerometersensor is granted.

Permission to use magnetometersensor is granted.

Permission to use gyroscopesensor is granted.

Device Orientation reads absolute = false alpha =null beta =null gamma =null

Mouse position (x,y) = (0,718) at = 1623348758411 ms

Mouse position (x,y) = (0,718) at = 1623348758412 ms

Mouse position (x,y) = (3,718) at = 1623348758416 ms

Mouse position (x,y) = (6,718) at = 1623348758424 ms

Mouse position (x,y) = (11,718) at = 1623348758432 ms

Mouse position (x,y) = (12,718) at = 1623348758448 ms

Mouse position (x,y) = (16,718) at = 1623348758448 ms

Mouse position (x,y) = (20,718) at = 1623348758470 ms

Mouse position (x,y) = (22,718) at = 1623348758478 ms

Mouse position (x,y) = (24,718) at = 1623348758486 ms

Mouse position (x,y) = (27,718) at = 1623348758494 ms

Mouse position (x,y) = (30,718) at = 1623348758502 ms

Mouse position (x,y) = (35,718) at = 1623348758510 ms

Mouse position (x,y) = (40,718) at = 1623348758518 ms

Mouse position (x,y) = (46,718) at = 1623348758532 ms

Mouse position (x,y) = (51,718) at = 1623348758540 ms

Mouse position (x,y) = (52,718) at = 1623348758556 ms

Mouse position (x,y) = (54,720) at = 1623348758564 ms

Mouse position (x,y) = (56,720) at = 1623348758586 ms

Mouse position (x,y) = (59,720) at = 1623348758602 ms

Mouse position (x,y) = (60,720) at = 1623348758696 ms

Mouse position (x,y) = (60,718) at = 1623348758710 ms

Latest click on: (x,y) = (60,718) at = 1623348758774 ms

Mouse position (x,y) = (60,716) at = 1623348758942 ms

Key pressed: Shift at time = 1623348760159ms

Key pressed: N at time = 1623348760311ms

N key was released at time = 1623348760379ms

Figura 5.1.- Plataforma de captura de datos vía web vista desde Google Chrome (izquierda), junto con la consola del navegador (derecha). URL: becaptcha.10x.es

- *read_json.py* Script de Python que procesa la base de datos obtenida con *getDatabase.py*, adecuando su formato para un correcto procesamiento de los datos.

El archivo *index.html* contiene los elementos básicos mencionados previamente para que el usuario interactúe con la web. En él se precargan los scripts necesarios para su funcionamiento, entre ellos el encargado de la captura de sensores, *readinfo.js*, así como scripts de Google para la subida de datos a Firebase y para la implementación de Google reCAPTCHA v3, que se detallarán en secciones posteriores. Además, desde este archivo se “llama” a las funciones de Javascript que arrancan la lectura de cada sensor y guardan los datos en Firebase, en tiempo real.

El fichero *readinfo.js* contiene las funciones implementadas en lenguaje Javascript para la extracción de sensores, así como la conexión con Google Firebase para la subida de datos.

Los archivos *index.html*, *readinfo.js* y *stylesheet.css* pueden verse en cualquier navegador accediendo a la consola e inspeccionando las fuentes, dado que es necesario su descarga por parte del navegador para la visualización y uso de la web.

5.2.2.- Detección del navegador y dispositivo

Como se mencionó previamente, se emplearán funciones incluidas en APIs de navegador para la obtención de datos de sensores. Habitualmente, existen diferencias en la forma en que cada navegador implementa o interpreta estas APIs, por ello es importante detectar el navegador empleado. La detección del navegador, así como el tipo de dispositivo (detectar si se trata de un dispositivo móvil o no) permite tomar decisiones a la hora de utilizar una API u otra, y asegurar que la captura de los datos se lleva a cabo con éxito en la mayoría de los casos.

Se implementan dos métodos para detectar qué navegador está usando el usuario: la técnica de *Duck Typing* y la lectura del *User Agent*.

- **Duck typing:** El *Duck Typing*, en programación, es una técnica empleada para determinar de qué tipo es un objeto mediante la observación de sus características.

«Si parece un pato, anda como un pato y hace ‘cuack’ como los patos, es un pato.»

Aplicado a la detección del navegador, se buscarán propiedades o atributos que solo se dan en un navegador determinado, para inferir que es éste el navegador que está usando el usuario.

- **User Agent:** El User Agent, en castellano Agente de Usuario, es una cadena de texto característica de los navegadores web, que permite identificar el protocolo de red, tipo de aplicación, sistema operativo, proveedor del software o la versión del software de la petición del agente de usuario. La lectura del User Agent se desaconseja ya que es fácil modificar esta cadena de texto por parte del usuario,

pero se utiliza como método secundario cuando el método de Duck Typing no consigue identificar el navegador. Además, permite obtener información adicional del dispositivo que podría resultar de utilidad.

Para la detección del navegador, se detectarán solo los navegadores más comunes o conocidos. En otro caso, se establecerá el navegador como “unknown” o desconocido. Los navegadores considerados son: Google Chrome, Mozilla Firefox, Opera, Microsoft Edge, Safari y Microsoft Internet Explorer.

Para detectar el tipo de dispositivo empleado se consideran las siguientes opciones:

- **Detección basada en el tamaño de la pantalla:** El valor del tamaño de la pantalla depende del valor que devuelve el navegador, y es habitualmente el tamaño de la ventana que tiene abierta. Esta técnica podría llevar a considerar erróneamente como dispositivo móvil un equipo con una ventana de navegador redimensionada a un tamaño pequeño.
- **Detección basada en detección de características:** Se basa en detectar si el usuario dispone de teclado, ratón o pantalla táctil y decidir en base a ello. Podría llevar a resultados erróneos, ya que existen equipos de escritorio o portátiles con pantalla táctil, así como los smartphones o tablets permiten la conexión de periféricos como teclados o ratones.
- **Detección basada en la lectura del User Agent:** La cadena de texto del User Agent contiene en ocasiones texto que identifica el tipo de dispositivo, aunque es fácilmente modificable, y por tanto, poco fiable.

Dado que no hay un método estandarizado para saber si un dispositivo es móvil o no, se emplea una combinación de las tres técnicas descritas, lo que permite llegar a conclusiones de forma más fiable.

Para llevar a cabo esta detección, se han desarrollado las siguientes funciones, dentro del archivo *readinfo.js*:

- *isMobile()* Determina si el usuario que ha visitado la web emplea o no un dispositivo móvil, basándose en las tres técnicas previamente descritas.

- *getBrowserDuckTyping()* Obtiene el navegador empleado por el método de Duck Typing.
- *getBrowserUserAgent()* Obtiene el navegador empleado por el método de lectura del User Agent.
- *getBrowser()* Combina las dos funciones anteriores para decidir desde qué navegador se está visitando la web, pero dando prioridad al método de Duck Typing.
- *getDeviceInfo()* Detecta el navegador, el User Agent y si el dispositivo es móvil o no, empleando las funciones anteriores, y almacena la información en la base de datos.

5.2.3.- Captura de sensores de los dispositivos móviles

Para la captura de los sensores de dispositivos móviles se emplean APIs web. La mayoría de las APIs de los distintos navegadores están documentadas en el sitio web de Mozilla Developers MDN Web Docs [11], como punto común de referencia para obtener información de las mismas. Para cada API, se describen métodos, propiedades o características, así como una tabla de compatibilidad entre los navegadores web más comunes.

Dado que se pretende capturar datos de sensores, se empleará la especificación *Generic Sensor API* [12], que define la manera de obtener datos de sensores en la web, de una forma consistente, y que pretende servir de estándar para la mayoría de navegadores web.

La captura de sensores del dispositivo está a menudo “protegida” en los navegadores, como medida de privacidad. Habitualmente, los navegadores muestran una ventana, indicándole al usuario que un sitio web quiere acceder a un sensor determinado, y solicitándole permiso de acceso al sensor. Por ello, se ha desarrollado la función *askPermissionAndRun()* en el archivo *readinfo.js*, que lanza una petición al navegador para acceder al sensor. Si no existe protección, el navegador devolverá el valor ‘granted’, permitiendo el acceso a los datos; Si existe, será el usuario quien acepte o deniegue el acceso al sensor, mediante la ventana indicada.

A continuación se describen los sensores capturados, las APIs empleadas para ello y su compatibilidad con los diferentes navegadores. La frecuencia de captura de muestras del Acelerómetro, Magnetómetro y Giroscopio puede definirse, siendo la frecuencia máxima disponible de 60Hz (limitación debida a la ejecución del código Javascript).

5.2.3.1.- Acelerómetro

Para registrar datos de acelerómetro se han empleado las API [13] *Accelerometer* y *LinearAccelerationSensor* de la *Generic Sensor API*. Con ellas se obtiene información de la aceleración aplicada a los ejes X,Y,Z del dispositivo que contiene el sensor. Esta aceleración está asociada a un sistema de coordenadas local, definido por el dispositivo.

Accelerometer: La aceleración es el cambio de velocidad de un dispositivo con respecto al tiempo, expresada m/s^2 [SI]. Se mide considerando la fuerza de la gravedad, por lo que un dispositivo en “caída libre” mediría una aceleración de $0m/s^2$. El sistema de coordenadas local empleado se muestra en la Figura 5.2. Se trata de un sistema de coordenadas cartesianas, (x,y,z) que está asociado al dispositivo físico. Para los dispositivos con pantalla, el sistema de coordenadas es el centro de la misma. Este sistema de coordenadas permanece fijo, independientemente de la orientación del dispositivo.

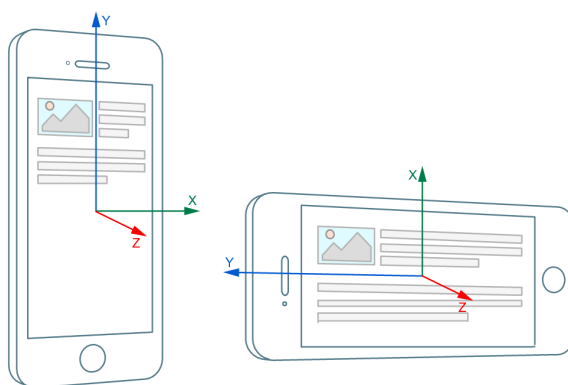


Figura 5.2.- Sistema de coordenadas local del sensor de acelerómetro. Fuente: [13]

LinearAccelerationSensor: La aceleración lineal es la aceleración aplicada al dispositivo que contiene el sensor, sin considerar la fuerza de la gravedad. Se considera una subclase del sensor anterior y emplea el mismo sistema de coordenadas.

Para la lectura de ambos sensores se emplea la función *accelerometer()* escrita en el fichero *readinfo.js*.

En la base de datos se registra el tipo de sensor de acelerómetro, las coordenadas X,Y,Z de aceleración registradas en m/s^2 y el timestamp, tiempo epoch en milisegundos en el que se registró la muestra.

5.2.3.2.- Giroscopio

El registro de datos de giroscopio se hace mediante la API *Gyroscope* [14] de la *Generic Sensor API*.

El sensor de giroscopio mide la velocidad angular en los ejes X,Y,Z. La velocidad angular es el ratio en el que un dispositivo rota alrededor de un determinado eje en un sistema de coordenadas local definido por éste, medido en radianes por segundo (rad/s). El signo depende de la dirección de rotación, empleándose la regla de la mano derecha por convenio, de forma que la rotación positiva es asociada a girar en el sentido de las agujas del reloj al rededor de un eje, cuando éste es visto a lo largo de la dirección positiva. La Figura 5.3 muestra el sistema de coordenadas empleado.

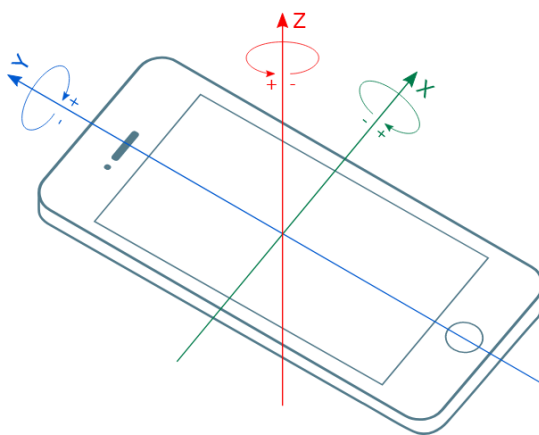


Figura 5.3.- Sistema de coordenadas local del sensor de giroscopio.Fuente: [14]

Para la lectura de datos del sensor se emplea la función *gyroscope()* escrita en el fichero *readinfo.js*.

En la base de datos se registran las coordenadas X,Y,Z de velocidad angular registradas en rad/s y el timestamp en milisegundos.

5.2.3.3.- Magnetómetro

El registro de datos del magnetómetro se hace mediante la API *Magnetometer* [17] de la *Generic Sensor API*. Para la lectura de datos del sensor se emplea la función *magnetometer()* escrita en el fichero *readinfo.js*.

Este sensor proporciona información del campo magnético detectado por el sensor del dispositivo, para los tres ejes (x,y,z) en micro Tesla. En la base de datos, se registra información para los ejes x,y,z, así como el timestamp asociado. El sistema de coordenadas de referencia en el dispositivo se muestra en la figura 5.4

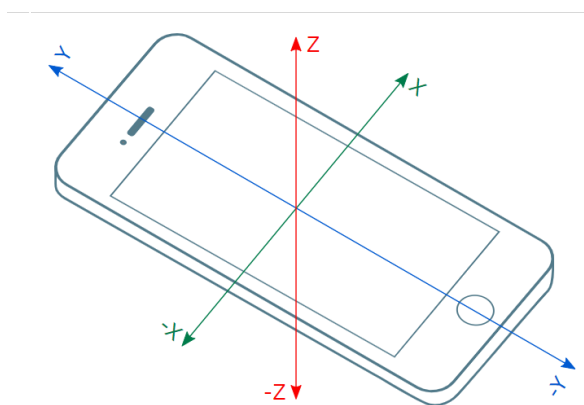


Figura 5.4.- Sistema de coordenadas local del sensor de magnetómetro. Fuente: [17]

5.2.3.4.- Orientación del dispositivo

El registro de datos de giroscopio se hace mediante la API *deviceorientation*, [15] a través de la función *handleOrientation()* implementada en el archivo *readinfo.js*.

La orientación del dispositivo proporciona información de la orientación física del dispositivo que entra en la página web. La información no es leída directamente de un sensor, sino que se combina la información de sensores como giroscopio, acelerómetro y brújula y se procesa a alto nivel para generar los datos.

El “sensor” proporciona los valores *alpha*, *beta*, *gamma* y *absolute*, que se registran en la base de datos junto con el tiempo de captura, o timestamp. El valor *absolute* es un dato de tipo booleano que indica si el dispositivo proporciona datos de orientación absoluta. Los valores *alpha*, *beta* y *gamma* expresan la orientación del dispositivo en términos de la transformación de una coordenada fija en el marco de coordenadas de la tierra, a una coordenada fija en el marco de coordenadas del dispositivo. [16]

Marco de coordenadas de la tierra: Su origen está en el centro de la tierra. Los ejes están alineados basados en la atracción de la gravedad y la orientación estándar del norte magnético.

- *Eje X:* A lo largo del plano de la tierra. Positivo hacia el este, negativo hacia el oeste.
- *Eje Y:* A lo largo del plano de la tierra. Positivo hacia el norte, negativo hacia el sur.
- *Eje Z:* Perpendicular al plano de la tierra, como si fuese una línea que une al dispositivo móvil con el centro de la tierra. Positivo alejándose del centro de la tierra, negativo acercándose.

Marco de coordenadas del dispositivo: Su origen se fija en el centro del dispositivo, tal y como muestra la Figura 5.5.

- *Eje x:* A lo largo del plano de la pantalla. Positivo hacia la derecha del dispositivo, negativo hacia la izquierda.
- *Eje y:* A lo largo del plano de la pantalla. Positivo hacia la parte superior del dispositivo, negativo hacia la parte inferior.
- *Eje z:* Perpendicular al plano de la pantalla, como si fuese una línea que une al dispositivo móvil con el centro de la tierra. Positivo saliendo de la pantalla, negativo entrando.

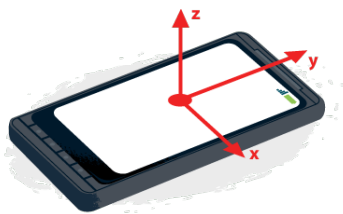


Figura 5.5.- Marco de coordenadas del dispositivo. Fuente: [15]

Los datos de orientación del dispositivo permiten registrar el movimiento del mismo, midiendo la rotación en cada eje, en términos del número de grados de diferencia entre el marco de coordenadas del dispositivo y el de la tierra, medido en grados.

- *alpha*: Representa el movimiento del dispositivo alrededor del eje z (rotación sobre sí mismo), expresado en grados, con valores en el rango $[0,360)$.
- *beta*: Representa el movimiento del dispositivo alrededor del eje x (movimiento adelante/atrás), expresado en grados, con valores en el rango $[-180,180)$.
- *gamma*: Representa el movimiento del dispositivo alrededor del eje y (movimiento izquierda/derecha), expresado en grados, con valores en el rango $[-90,90)$.

La Figura 5.6 describe los movimientos asociados a los ángulos *alpha*, *beta* y *gamma*.

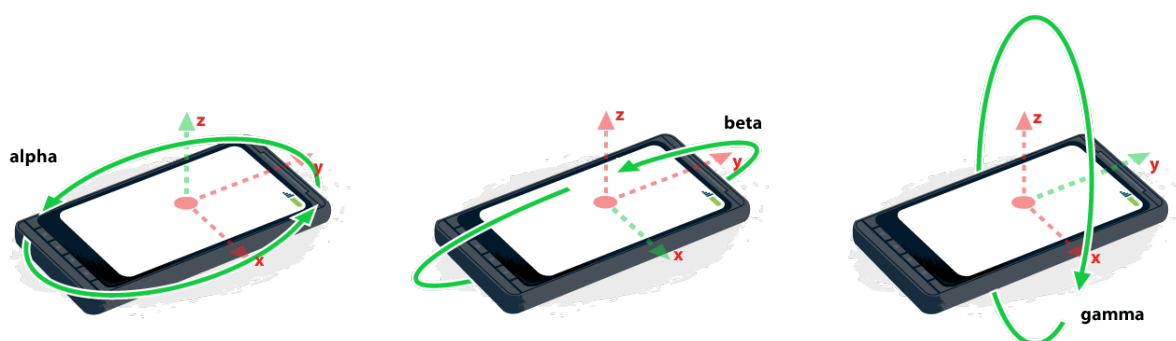


Figura 5.6.- Movimientos asociados a los ángulos *alpha*, *beta* y *gamma* en un dispositivo móvil. Fuente: [15]

5.2.3.5.- Otros sensores implementados

Se han implementado sensores como el sensor de luz, y sensor de proximidad empleando para ello las APIs *AmbientLightSensor*, *deviceproximity* y *userproximity* existentes en la documentación de MDW Web Docs. [11] Sin embargo, aunque existen estas APIs, son todavía experimentales, y no tienen compatibilidad con los navegadores más empleados. En algunos casos, es necesario modificar propiedades en los navegadores para permitir la lectura de estos sensores. Es el caso de Mozilla Firefox, que bloquea por defecto la lectura del sensor de luz, aunque este puede habilitarse cambiando la configuración del navegador mediante la dirección *about:config*. Dado que se pretende capturar datos para cualquier usuario que entre en la web, no sería posible obtener datos de estos sensores sin forzar al usuario a configurar su navegador, por tanto, se ha optado por excluir estos sensores de la base de datos de Google Firebase.

5.2.4.- Captura de teclado, ratón y pantallas táctiles

La captura de este tipo de interacciones está basada en eventos. Los eventos son acciones que ocurren en el sistema, tras los cuales suele ejecutarse una función que puede programarse (habitualmente en Javascript). Los eventos de la web suelen estar asociados a la ventana, o bien a elementos dentro del código HTML. Un ejemplo de evento puede ser cuando el usuario pulsa una tecla, o cuando la web termina de cargarse.

Para la captura de teclado, ratón y pantallas táctiles, se desarrollan funciones que se ejecutarán tras determinados eventos.

5.2.4.1.- Teclado

La captura de las pulsaciones de teclado se realiza mediante los eventos *keyup* y *keydown* en HTML. Cada vez que se detectan estos eventos en el navegador, se hace una llamada a las funciones *keyUp()* y *keyDown()* respectivamente, implementadas en el fichero *readinfo.js*.

- *keyDown()* Registra las teclas que se pulsán, en el momento de presionarlas. Incluye combinaciones de Ctrl+Tecla.
- *keyUp()* Registra el momento en el que se suelta la tecla, y se deja de pulsar.

En la base de datos se guarda el tipo de evento (*keyDown* o *keyUp*), el timestamp en milisegundos y las teclas pulsadas o soltadas.

5.2.4.2.- Ratón (Mouse)

Para capturar el movimiento del ratón por la pantalla se utilizan los eventos *mousemove*, *mousedown* y *mouseleave* de HTML. Las funciones que gestionan estos eventos, implementadas en *readinfo.js*, se describen a continuación:

- *mouseMove()* Obtiene la posición x,y de la ubicación del puntero del mouse en la pantalla, y se actualiza con el movimiento del ratón.
- *mouseDown()* Registra los eventos de presionar el botón izquierdo del ratón sobre la página web. Se diferencia del evento de “click” en que este último solo se registra al presionar y soltar de nuevo.
- *mouseLeave()* Registra en qué posición (x,y), el ratón ha abandonado la ventana. Este valor no se registra en la base de datos.

La coordenada x registra valores entre 0 y la anchura máxima de la ventana del navegador, mientras que la coordenada y registra valores entre 0 y la altura máxima de la ventana del navegador. El origen de coordenadas se encuentra en la esquina superior izquierda de la ventana del navegador. En la base de datos se guarda el tipo de evento (Move o Click), el timestamp en milisegundos y las coordenadas (x,y) asociadas.

5.2.4.3.- Pantalla táctil (Touchscreen)

La interacción con la pantalla táctil se gestiona mediante los eventos *touchstart*, *touchmove*, *touchend* y *touchcancel* de HTML. Aunque ha resultado la interacción más difícil de implementar, se ha conseguido registrar eventos de toque en la pantalla, siguiendo el movimiento que se describe hasta soltar el dedo de la misma. Además,

se registran eventos multitoque, de forma que si se toca la pantalla con varios dedos distintos y se mueven los mismos por la pantalla, los movimientos de cada toque serían registrados de forma independiente, permitiendo seguir las “trazas” de cada uno de ellos de forma independiente.

El seguimiento de los toques en la pantalla se ha implementado distinguiendo entre toques absolutos y relativos. Para identificar cada toque se utilizan identificadores o IDs.

- *Toques absolutos*: Se registran con el identificador *touchID_abs* en la base de datos. Es el numero de toque desde que se cargó la página web.
- *Toques relativos*: Se registra con el identificador *touchID_rel* en la base de datos. Es el número de toques simultáneos que se registran durante un toque absoluto. De esta forma, cada toque absoluto puede constar de varios toques simultáneos.

Por ejemplo, si se toca la pantalla con dos dedos simultáneamente, se registraría un nuevo toque absoluto, que contendría (bajo el mismo ID absoluto) dos toques relativos (uno relativo a cada dedo).

Las funciones que gestionan estos eventos, implementadas en *readinfo.js*, se describen a continuación, junto con funciones auxiliares necesarias:

- *touchStart()* Registra nuevos toques, las coordenadas (x,y) iniciales de los mismos y el número de toques simultáneos, así como el timestamp asociado.
- *touchMove()* Registra las coordenadas (x,y) por las que se desplaza el “toque” por la pantalla.
- *touchEnd()* Gestiona el fin de cada toque, y registra las coordenadas del último punto asociado al toque.
- *touchCancel()* Gestiona la cancelación de toques. Ocurre, por ejemplo, si el usuario arrastra el dedo hasta salirse de la ventana del navegador.
- *copyTouch()* Función auxiliar que permite identificar cada toque para poder “seguirlo”.
- *ongoingTouchIndexById()* Función auxiliar que permite gestionar el seguimiento de los toques, así como eliminar los toques terminados.

En resumen, la interacción táctil se registra en la base de datos como las coordenadas en la pantalla iniciales, intermedias, y finales que registra cada toque en la misma, así como los timestamps asociados, permitiendo distinguir entre toques en el tiempo, mediante IDs absolutos, y toques simultáneos, mediante IDs relativos.

5.2.5.- Implementación de Google reCAPTCHA v3

Google reCAPTCHA [20] es un servicio de Google que permite integrar un sistema CAPTCHA en un sitio web. En la versión 3, en lugar de mostrar un “desafío” de CAPTCHA para que el usuario interactúe, se devuelve una puntuación en función de la interacción del usuario con la página web. Esto se realiza en segundo plano, de forma totalmente transparente al usuario. El estudio de la interacción se realiza cargando un script de Google en la cabecera `<head>` del archivo *index.html*.

La integración de reCAPTCHA v3 puede hacerse de diferentes formas. En el proyecto, este servicio se ha implementado para ejecutarse al hacer click en el botón “Get score” de la página web. Este botón llama al script *recaptchav3_control.php*, que se ejecuta en el servidor (en el caso del proyecto, la Raspberry Pi 3B+ que aloja la página web), y lanza una petición mediante el método POST del protocolo HTTP a un servidor de Google. En dicha petición se envían parámetros que identifican la web, como una clave secreta (SECRET KEY) y la clave de sitio web (SITE KEY), que deben asociarse previamente a una cuenta de Google. Tras la petición, se obtiene una respuesta en el código PHP implementado, en formato JSON. Esta respuesta, que está en el servidor, debe enviarse de nuevo al cliente (navegador del usuario que entra en el sitio web).

Para implementar este intercambio de datos se ha optado por un método sencillo, que consiste en utilizar el elemento `<iframe>` de HTML. Este elemento permite incrustar una página web dentro de otra. Por ello, el código PHP se ejecutará dentro de un `<iframe>` oculto al usuario. Al ejecutarse el PHP, se imprime mediante el comando *echo* la respuesta que envía el servidor de Google dentro del `<iframe>`, y se utiliza la función de Javascript *getScoreFromIframe()* implementada en *readinfo.js* para extraer

la información y guardarla en la base de datos. La Figura 5.7 muestra el formato de la respuesta que se obtiene empleando reCAPTCHA v3 de Google.

```
{
  "success": true,
  "challenge_ts": "2021-04-28T18:44:17Z",
  "hostname": "tfmtest.10x.es",
  "score": 0.9,
  "action": "test"
}
```

Figura 5.7.- Respuesta en formato JSON que devuelve la petición de reCAPTCHA v3.

En la base de datos se registrarán los parámetros *Success* y *Score*. El parámetro *Success* será *True* si se considera que la interacción la produjo una persona, y *False* en el caso de un bot. Por otro lado, el parámetro *Score* asigna una puntuación entre 0 y 1, siendo 0 la mayor probabilidad de que la interacción la produjese un bot, y 1 la mayor probabilidad de que la produjese una persona. En una implementación real de esta tecnología, el desarrollador web escogería un umbral a partir del cual considerar una interacción como generada por un bot, en función del *Score* obtenido.

5.2.6.- Implementación de base de datos en tiempo real con Google Firebase

Para guardar los datos capturados de la interacción de los usuarios con la página web, se ha optado por utilizar el servicio de base de datos en tiempo real de Google Firebase: RealTime Database (Google RTD). Dado que el código Javascript se ejecuta en el navegador web del usuario que visita la página, se ha considerado más sencillo enviar estos datos a una base de datos en tiempo real, en lugar de crear una base de datos en el servidor. Si se desea disponer de los datos en el servidor, éstos son fácilmente accesibles mediante la ejecución del script implementado en Python *getDatabase.py*.

La base de datos se ha alojado en el sitio web <https://becaptchaweb.firebaseio.com/>.

Para la integración de la base de datos con la plataforma de captura es necesario incluir los siguientes scripts de Google en la cabecera del documento HTML:


```
<script src="https://www.gstatic.com/firebasejs/6.2.0/firebase-app.js"></script>  
<script src="https://www.gstatic.com/firebasejs/6.2.0/firebase-database.js"></script>
```

Además, se han implementado las siguientes funciones en el archivo *readinfo.js*:

- *loadFirebase()* Se llama a esta función al cargar la web. Permite inicializar la base de datos, enviando una variable con los parámetros de configuración a los servidores de Google.
- *pushToFirebase()* Función implementada para guardar los datos en la base de datos de manera sencilla desde el código que gestiona cada dato o sensor.

La base de datos guarda información relativa a cada conexión que se hace a la página web, identificando cada conexión mediante el timestamp del momento en que el usuario se conecta, haciendo uso de la función *getUserID()* implementada en *readinfo.js*. Esto permite distinguir varias conexiones de un mismo usuario. A su vez, cada vez que se registran datos, estos se guardan con un identificador generado por Google RTD. Los datos se guardan en formato JSON.

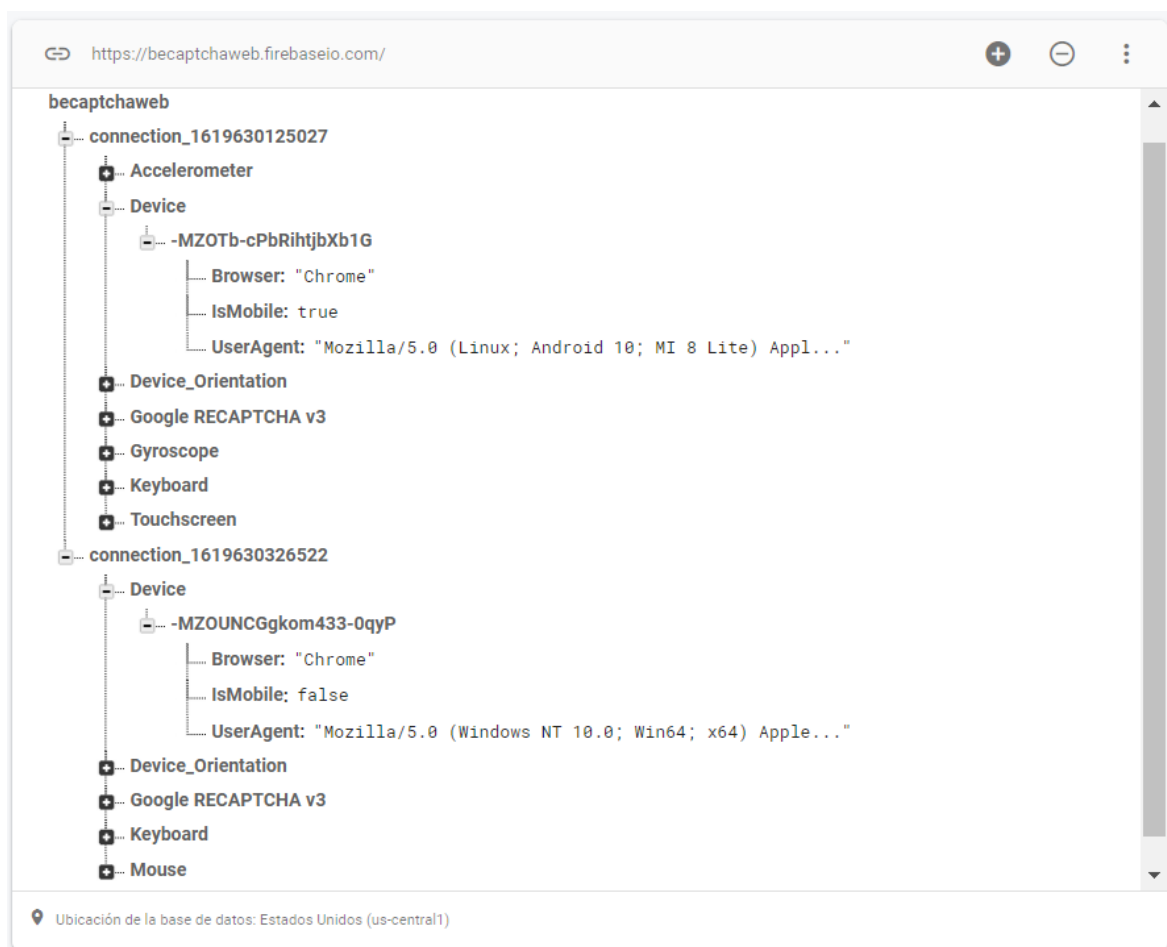


Figura 5.8.- Ejemplo de dos conexiones realizadas a la página web, desde diferentes dispositivos, visto desde la herramienta de gestión de Google Realtime Database.

La Figura 5.8 muestra un ejemplo de dos conexiones realizadas a una base de datos, la primera mediante un dispositivo móvil Android (Xiaomi Mi 8 Lite), y la segunda mediante un equipo de sobremesa (Windows 10). En ambos casos se ha utilizado el navegador Chrome. Puede observarse el tipo de datos que se almacena en cada conexión. La información del dispositivo, almacenada en "Device" consigue identificar correctamente el navegador desde el que se accede, si el dispositivo es móvil o no, e incluso puede identificarse el sistema operativo mediante el User Agent correctamente.

Cabe destacar que la captura y registro de los datos se realiza durante cinco minutos, tras los cuales se dejan de registrar datos. Esto evita que un usuario que deje la web abierta o interactúe con ella durante mucho tiempo pueda saturar la base de datos.

5.2.6.1.- Procesamiento de la base de datos con Python

Alternativamente, se pueden obtener los datos desde cualquier equipo a través de los scripts desarrollados en Python. Los archivos *getDb.py* y *read_json.py* permiten descargar y procesar la base de datos, de forma que se eliminan los ID que intercala Google RTD en cada guardado de datos, dado que no serán necesarios. La Figura 5.9 muestra la apariencia de la base de datos cargada desde Python, tras su limpieza.

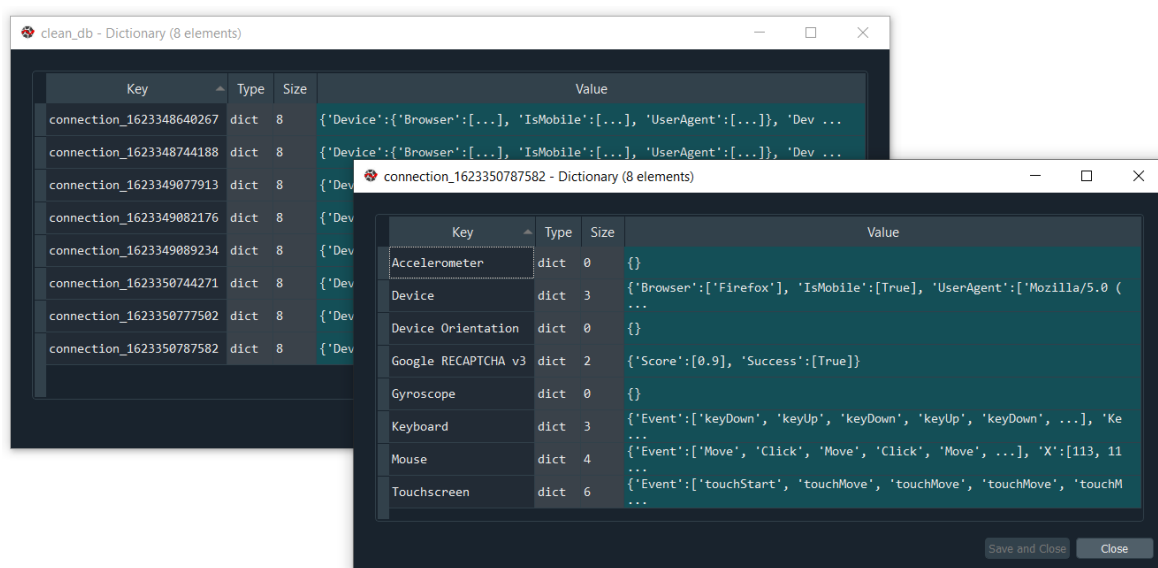


Figura 5.9.- Aspecto de la base de datos obtenida tras el postprocesado con Python.

La descarga de la base de datos empleando código Python permitirá en un futuro estudiar los datos capturados, así como entrenar modelos de detección de bots en base a los mismos.

5.3.- Rendimiento de la solución implementada y limitaciones conocidas

Para comprobar el correcto funcionamiento de la plataforma de captura de datos se han empleado varios navegadores web, con los que se ha accedido a la página empleando tanto equipos de sobremesa como dispositivos móviles. La Tabla 5.2 muestra un resumen de la compatibilidad de las APIs implementadas con cada uno de los navegadores.

	Chrome	Edge	Firefox	IE	Opera	Safari	Webview Android	Chrome Android	Firefox Android	Opera Android	Safari iOS
Acelerómetro	✓	✓	✗	✗	✓	✗	✓	✓	✗	✓	✗
Giroscopio	✓	✓	✗	✗	✓	✗	✓	✓	✗	✓	✗
Magnetómetro	✓	✓	✗	✗	✓	✗	✗	✓	✗	✓	✗
Orientación	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
Sensor luz	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Sensor Proximidad	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Teclado	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Ratón	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pantalla Táctil	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Tabla 5.2.- Compatibilidad de las APIs implementadas con los navegadores más utilizados a nivel mundial, para navegadores convencionales y navegadores para dispositivos móviles.

Los navegadores con mejores resultados son Google Chrome (desktop y Android), Edge y Opera (desktop y Android).

Como limitaciones, se ha observado que en algunos casos el sistema falla a la hora de obtener las pulsaciones de teclado en dispositivos Android, registrando las teclas con un valor desconocido “undefined” o “unknown”. Esto podría deberse a que el teclado es gestionado por una aplicación externa al navegador en estos sistemas, que no permite leer los datos correctamente. Por otro lado, el sensor de magnetómetro deniega el acceso al mismo para algunos navegadores, siendo necesario habilitarlo desde la configuración

del navegador. No obstante, se considera que la plataforma de captura permite obtener los datos de forma satisfactoria.

6. Desarrollo de un demostrador web con Flask

Para mostrar la utilidad y la aplicabilidad del trabajo propuesto se ha desarrollado un demostrador web. El demostrador pretende mostrar de qué manera sería posible integrar el trabajo realizado y emplearlo en aplicaciones reales.

El demostrador es una aplicación web desarrollada con Flask [18]. Flask es un framework web que sigue el estándar *Web Server Gateway Interface* o WSGI. Este estándar define el modo en que un servidor web se comunica con aplicaciones web mediante el uso de Python. El uso de este framework permite integrar el código Python previamente desarrollado de una forma sencilla. Además, se emplean otras tecnologías web como Javascript, HTML, CSS, y se mantiene la interacción con Google Real Time Database previamente realizada.

6.1.- Descripción de la aplicación web

El objetivo de la web es registrar la interacción de un usuario al navegar por la misma, y determinar si éste se trata de un humano, o de un software automatizado o bot. La web está centrada en detectar interacciones desde dispositivos móviles, dado que se basa en registrar eventos de interacción táctil y lecturas de acelerómetro del dispositivo. Sin embargo, incorpora la captura de todos los datos que registra la plataforma de captura de datos descrita en la el capítulo 5.

La Figura 6.1 muestra capturas de la página principal de la aplicación, la cual se divide en tres secciones:

- Interacción táctil
- Acelerómetro
- Ambos sensores



Figura 6.1.- Página principal de la aplicación desarrollada: BeCaptchaWeb.

En cada una de las secciones se permite al usuario interactuar con las redes neuronales entrenadas para el proyecto, realizando predicciones sobre las mismas para generar secuencias artificiales tanto de interacción táctil como de aceleración, empleando para ello los generadores previamente entrenados con las GAN. Las secuencias de entrada se basan en ruido aleatorio gaussiano.

En el caso de la interacción táctil, se permite además registrar la interacción con la pantalla táctil al realizar un movimiento deslizando el dedo hacia la derecha sobre un cuadro, lo que a su vez lanzará la captura del sensor de aceleración durante el movimiento, obteniéndose la secuencia de aceleración asociada. Esta secuencia de aceleración podrá verse posteriormente en la sección correspondiente. El cuadro mencionado se muestra en la Figura 6.2.

Desliza el dedo hacia la derecha para registrar un movimiento.



La lectura del acelerómetro registrada durante el movimiento puede verse en la sección "Acelerómetro".

Figura 6.2.- Cuadro para la captura de secuencias de interacción táctil, junto con la secuencia de aceleración asociada.

Tanto las secuencias generadas por el usuario como las generadas artificialmente por la GAN se muestran por pantalla al usuario en el momento de su generación (tras pulsar el botón asociado). Además, dichas secuencias se “pasan” como entrada a los dos detectores de bots desarrollados:

- **Discriminador de la red GAN:** Permite evaluar la capacidad de detección de bots del discriminador desarrollado.
- **Clasificador entre secuencias generadas por la GAN y las reales de HuMIdB:** Permite evaluar la capacidad de detección de bots del clasificador desarrollado.

Cada uno de los clasificadores devuelve la puntuación o *score* asociado a la secuencia de entrada. Para determinar si la interacción fue producida por un humano o un bot, se compara el *score* obtenido con los umbrales determinados para cada clasificador en el capítulo 4. Finalmente se muestran los resultados al usuario, tal y como se observa en la Figura 6.3 (a).

En el caso de generar secuencias a partir de ruido aleatorio, se evalúa la capacidad de generación de muestras realistas del mismo, empleando para ello el clasificador entre ruido y secuencias reales de HuMIdB entrenado. El objetivo de este clasificador es determinar si las secuencias generadas por la red GAN están lo suficientemente alejadas del ruido, y, a su vez, si estas son similares a las observadas en la base de datos de HuMIdB. En definitiva, comprobar la calidad de las secuencias generadas por la GAN.

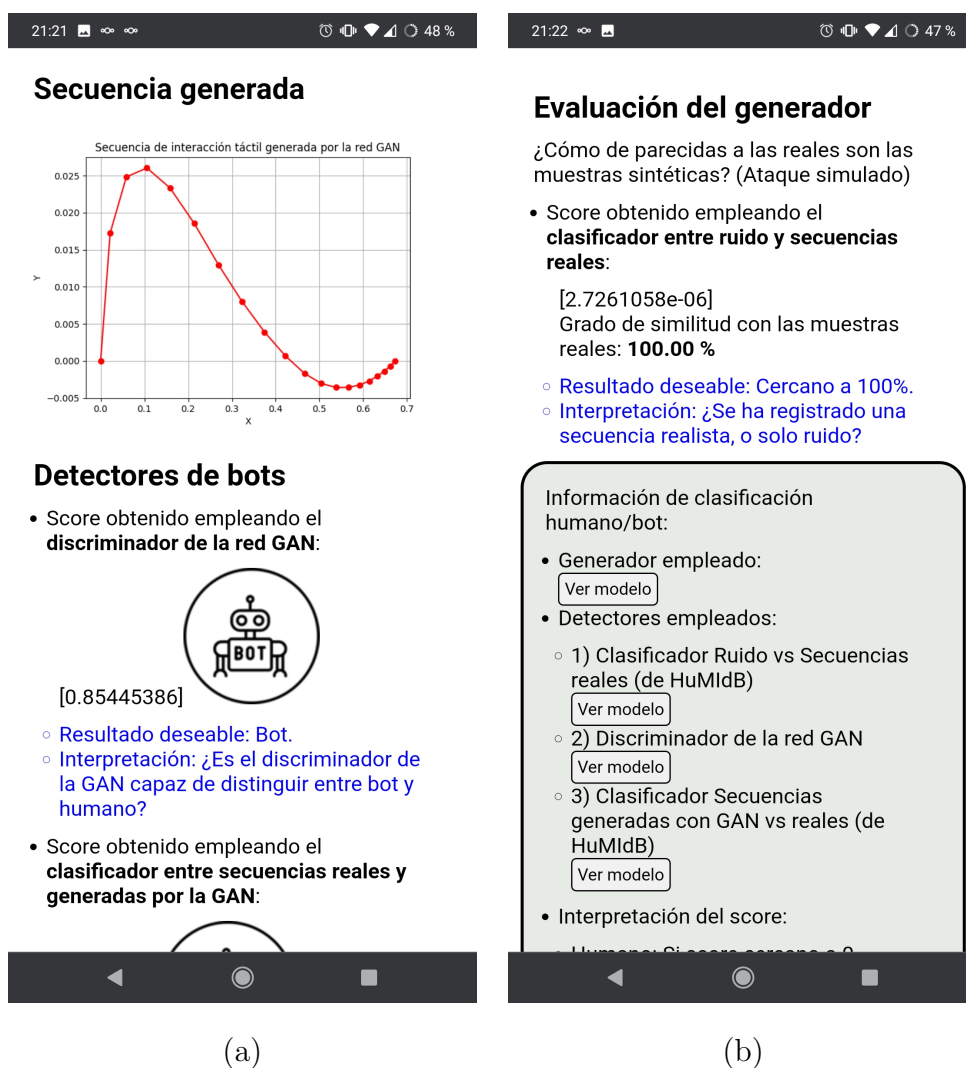


Figura 6.3.- Resultados tras generar una secuencia de interacción táctil a partir de ruido aleatorio. (a) Secuencia generada y resultados de detección de bots. (b) Evaluación de los resultados del generador e información adicional.

Adicionalmente, se muestra información gráfica de las capas que componen cada modelo de red neuronal empleado y su interconexión (Figura 6.3 (b)).

6.2.- Conclusiones

El demostrador desarrollado permite integrar la plataforma de captura de datos con los modelos de redes neuronales en una sola aplicación. Los datos capturados pueden emplearse para realizar predicciones sobre las redes neuronales. De esta forma,

la interacción con el dispositivo es registrada y procesada mientras el usuario navega por la web, pudiendo determinarse si éste se trata de un humano o un bot.

Es importante tener en cuenta que los modelos de redes neuronales han sido entrenados con datos procedentes de la base de datos de HuMidB. Estos datos fueron registrados a través de una aplicación de Android, empleando para ello lenguaje Java. Sin embargo, los datos capturados mediante la plataforma realizada se obtienen empleando Javascript. Al emplearse información de entrenamiento de una fuente distinta a la información a evaluar, el rendimiento de los detectores no es el óptimo, siendo más difícil la correcta detección de bots.

En un escenario óptimo, sería necesario emplear la plataforma de captura para registrar datos de muchos usuarios durante un tiempo, para posteriormente procesarlos y entrenar las redes neuronales con ellos. Sin embargo, este proceso es largo y requiere de colaboración, por lo que se deja como línea futura, y no se implementa en este proyecto.

Aun así, los resultados obtenidos al realizar pruebas con la aplicación web son satisfactorios. Se observa que las secuencias generadas en la web y las capturadas en la base de datos de HuMidB son similares, por lo que los detectores consiguen buenos resultados.

6.3.- Subida de la aplicación a Github y Azure Web Services

Para permitir el acceso a la web desde cualquier lugar de Internet, se ha alojado la misma en un servidor de Microsoft Azure, conectándola para ello con un repositorio en GitHub [19].

Microsoft Azure es un servicio de computación en la nube, que pone a disposición de los clientes diferentes servicios de software, infraestructura y plataforma. Permite construir, probar, desplegar y administrar aplicaciones y servicios mediante el uso de sus centros de datos. A continuación se describe el proceso seguido para el despliegue de la aplicación en el servidor.

Una vez se dispone de la aplicación desarrollada en Flask, se procede a su alojamiento en un servidor de Azure. Para ello, es necesario disponer de una suscripción de Azure. Se ha empleado la suscripción “Azure para estudiantes” incluida en la cuenta de estudiante de la universidad.

En primer lugar, es necesario crear un **grupo de recursos** de Azure. Los grupos de recursos son contenedores lógicos en los cuales se despliegan servicios de Azure, tales como la propia aplicación web a desplegar. Para ello, se escoge “BeCaptchaWeb” como nombre del grupo de recursos, y se establece el servidor en la región “Oeste de Europa”.

Una vez creado, se creará una **aplicación web**, dentro del grupo de recursos. Para ello, se realiza la configuración que se muestra en la Figura 6.4.

Detalles del proyecto

Seleccione una suscripción para administrar los recursos implementados y los costos. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción * ⓘ Azure para estudiantes ▼

Grupo de recursos * ⓘ BeCaptchaWeb ▼

[Crear nuevo](#)

Detalles de instancia

Nombre * becaptchaweb ✓
.azurewebsites.net

Publicar * ☒ Código ☐ Contenedor de Docker

Pila del entorno en tiempo de ejecución * Python 3.8 ▼

Sistema operativo * ☒ Linux ☐ Windows

Región * West Europe ▼

[¿No encuentra su plan de App Service? Pruebe otra región.](#)

Figura 6.4.- Configuración de la aplicación web desde Azure.

Por último, es necesario crear un **plan de servicio de App**, que define el conjunto de recursos que se emplearán para alojar y ejecutar la aplicación web. Se elige un plan de servicio básico B1, que se trata de una máquina con los siguientes recursos:

- Núcleos: 1

- RAM: 1,75 GB
- Almacenamiento: 10 GB
- Sistema Operativo: Linux Ubuntu 20.04 LTS

Una vez el servicio de Azure está configurado, es necesario enviar el código de la aplicación en Flask para su despliegue. Para simplificar el proceso se ha escogido emplear un repositorio de Github, que deberá contener la aplicación web y todos los archivos asociados, tal como muestra la Figura 6.5.

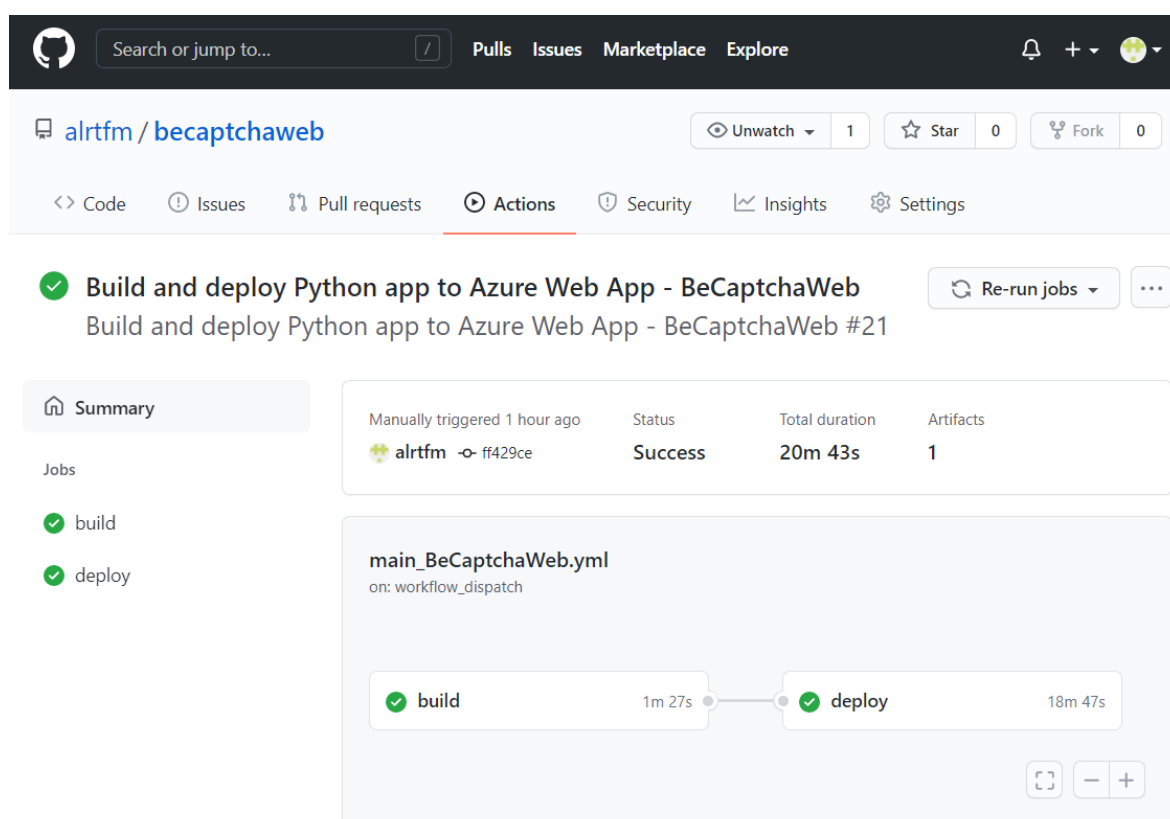
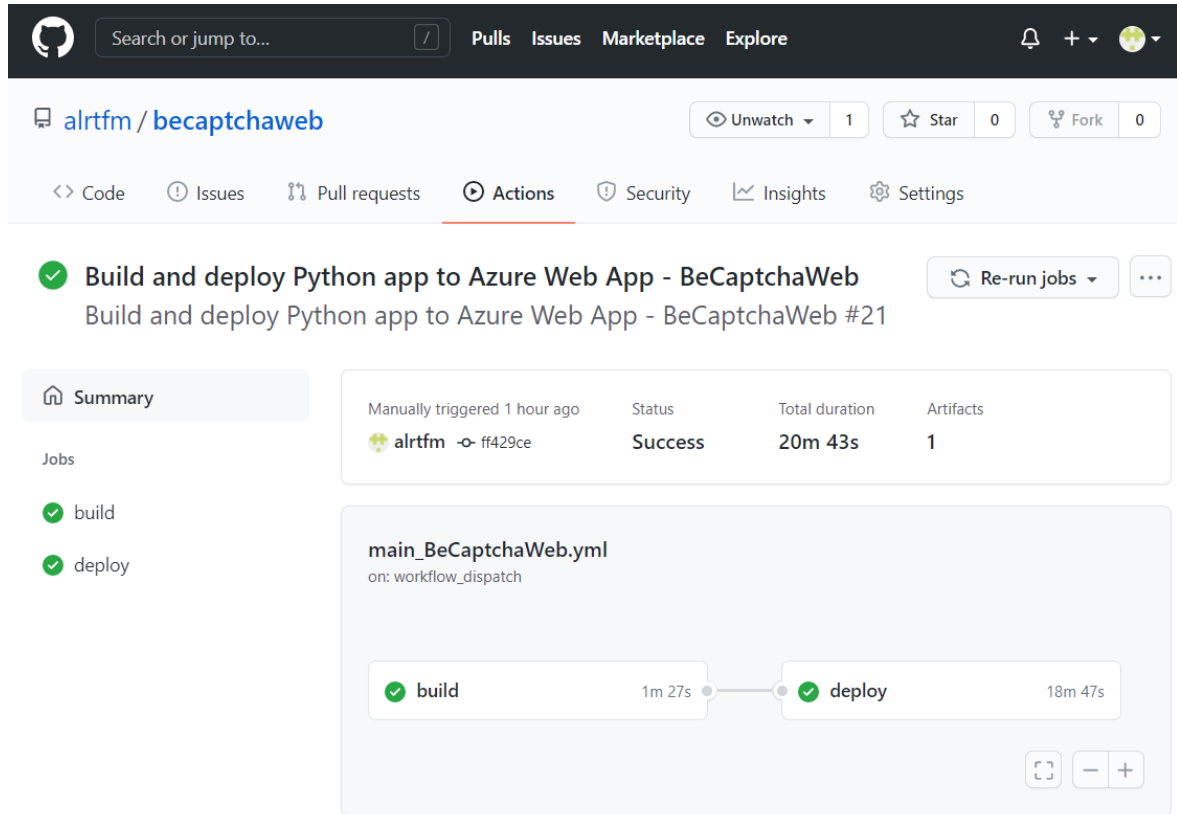


Figura 6.5.- Repositorio de Github con la aplicación del proyecto. URL: <https://github.com/alrtfm/becapthaweb>

Desde el **Centro de Implementación** disponible en Azure, se inicia sesión con la cuenta de Github y se escoge el repositorio que contendrá la aplicación. Una vez vinculado, se crea en el repositorio de Github un archivo de extensión *yml* en la carpeta *.github/workflows*, que contiene el proceso a seguir para realizar el despliegue de la aplicación en la máquina de Azure. Desde la pestaña Actions de Github, puede lanzarse

el proceso, que una vez completado permitirá usar la aplicación web accediendo a la URL correspondiente.



The screenshot shows the GitHub Actions interface for the repository `alrtfm/becaptchaweb`. The workflow `Build and deploy Python app to Azure Web App - BeCaptchaWeb` is displayed, showing it was manually triggered 1 hour ago and completed successfully. The workflow consists of two jobs: `build` (1m 27s) and `deploy` (18m 47s).

Manually triggered	Status	Total duration	Artifacts
1 hour ago	Success	20m 43s	1

The workflow `main_BeCaptchaWeb.yml` is triggered on `workflow_dispatch`. The jobs are:

- `build`: 1m 27s
- `deploy`: 18m 47s

Figura 6.6.- Despliegue de la aplicación en Azure Web Services desde el repositorio GitHub

7. Conclusiones y líneas futuras

7.1.- Conclusiones

La realización del proyecto ha servido para comprender en mayor profundidad los procesos a llevar a cabo cuando se trabaja en el campo del aprendizaje automático. Durante el proyecto se ha trabajado no solamente en el análisis de datos y el entrenamiento de modelos de *machine learning*, sino que también se ha realizado una plataforma que permite capturar datos para crear una base de datos propia. Además, lo desarrollado se pone en práctica mediante el desarrollo de un demostrador, lo cual permite tener una visión de la aplicación real del proyecto.

En primer lugar se realizó un estudio de una base de datos existente. Comprender qué métodos se han llevado a cabo para la captura de los datos ha permitido aplicar un procesamiento adecuado a los mismos. El tratamiento de los datos antes de introducirlos a un modelo para su entrenamiento resulta fundamental para obtener resultados válidos. El análisis estadístico de los datos ha permitido tomar decisiones a la hora de construir los modelos de redes neuronales empleados en el proyecto.

Desde el punto de vista de la aplicabilidad, se han conseguido desarrollar modelos válidos de generación de secuencias, tanto para aceleración como para interacción táctil, empleando distintos generadores. Este proceso permite simular los ataques que serían realizados por programas automatizados. En base a estos ataques, se han desarrollado métodos de detección de bots, tanto empleando los discriminadores de las redes GAN como con clasificadores basados en redes neuronales “tradicionales”.

En cuanto al estudio de la correlación entre secuencias de aceleración e interacción táctil, se ha realizado tanto un análisis estadístico, como un análisis mediante el empleo de redes neuronales. Ambos han llevado a conclusiones ligeramente diferentes, por lo que no se ha podido determinar con rotundidad si existe una correlación real. Sin embargo, los resultados analizando los datos con redes neuronales sugieren una ligera correlación, por lo que no se ha descartado que exista. Debido a esto, se ha tratado

de desarrollar un modelo conjunto de red GAN, que permita tanto generación como discriminación de secuencias de manera conjunta. Se han visto diversas limitaciones a la hora de desarrollar los modelos, que han llevado a secuencias generadas poco realistas, por lo que no se considera que se haya logrado el objetivo de generación y discriminación conjunta.

Por otro lado, el desarrollo de la plataforma de captura ha permitido conocer cuáles son las limitaciones a la hora de capturar datos de los sensores de diferentes dispositivos vía web. La interpretación de los códigos por parte de los diferentes navegadores es distinta, y lleva en ocasiones a problemas de compatibilidad. Aunque se han usado APIs que pretenden implantarse como estándares, como la *Generic Sensor API*, todavía queda mucho trabajo por hacer para estandarizar la web.

Por último, el desarrollo de una aplicación web que integra la plataforma de captura realizada con los modelos de redes neuronales entrenados ha permitido conocer cómo implementar lo desarrollado. Se han visto qué tecnologías son necesarias, y de qué forma se deben integrar para permitir el uso de las mismas en una aplicación real.

7.2.- Líneas futuras de investigación

Sería interesante continuar el proyecto incorporando el estudio de más sensores, y observando la correlación que pueda existir entre las secuencias capturadas por los mismos en el mismo instante temporal, así como conseguir el desarrollar modelos que permitan la generación y discriminación de secuencias de manera conjunta.

La plataforma de captura puede mejorarse incorporando métodos de captura de sensores que permitan adaptarse a distintos navegadores, de forma que se eliminen ciertas limitaciones debidas a incompatibilidades por la elección del navegador. Además, sería conveniente emplear la plataforma web desarrollada para la creación de una base de datos que contenga los datos capturados a través de ésta. Esto permitiría caracterizar las interacciones entre las personas y los navegadores web, para entrenar modelos que se empleen específicamente como medida de seguridad en servicios web.



Bibliografía

- [1] HuMIdb database(Human Mobile Interaction database) <https://github.com/BiDALab/HuMIdb>
- [2] A. Acien, A. Morales, J. Fierrez, R. Vera Rodriguez, O. Delgado Mohatar, *BeCAPTCHA: Bot Detection in Smartphone Interaction using Touchscreen Biometrics and Mobile Sensors*, Engineering Applications of Artificial Intelligence (to appear), Elsevier, 2020.
- [3] A. Acien, A. Morales, J. Fierrez, R. Vera Rodriguez, Ivan Bartolome, *BeCAPTCHA: Detecting Human Behavior in Smartphone Interaction using Multiple Inbuilt Sensors* "Sensors", In AA AI Workshop on Artificial for Cyber Security (AICS), Feb. 2020.
- [4] Chollet, Francois, *Deep Learning with Python*, New York, NY: Manning Publications.
- [5] Ian J. Goodfellow, Jean Pouget Abadie, Mehdi Mirza, Bing Xu, David Warde Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative adversarial nets* In Proceedings of the 27th International Conference on Neural Information Processing Systems Volume 2 (NIPS'14). MIT Press, Cambridge, MA, USA, 2672 2680. 2014
- [6] Brodić, D. and Amelio, A, *The CAPTCHA: Perspectives and Challenges: Perspectives and Challenges in Artificial Intelligence*, (Smart Innovation, Systems and Technologies), (Volume 162), Springer International Publishing, Cham.
- [7] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. *The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors*. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). Association for Computing Machinery, New York, NY, USA, 1515 1532.

- [8] *Mobile Browser Market Share Worldwide* – StatCounter Global Stats.
<https://gs.statcounter.com/browser-market-share/mobile/worldwide/#monthly-202011-202011-bar>
- [9] NetMarketshare, *Browser market share* <https://netmarketshare.com>
- [10] *Introducción a las APIs web*, MDN Web Docs, https://developer.mozilla.org/es/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
- [11] *Referencia de la API Web*, MDN Web Docs <https://developer.mozilla.org/es/docs/Web/API>
- [12] *Generic Sensor API*, W3C Candidate Recommendation, 12 December 2019
<https://www.w3.org/TR/generic-sensor/>
- [13] *Accelerometer Interface - Generic Sensor API* <https://w3c.github.io/accelerometer/>
- [14] *Gyroscope Interface - Generic Sensor API* <https://w3c.github.io/gyroscope/>
- [15] *DeviceOrientation Event Specification - Generic Sensor API* <https://w3c.github.io/deviceorientation/>
- [16] *Orientation and motion data explained - MDN Web Docs* https://developer.mozilla.org/en-US/docs/Web/Events/Orientation_and_motion_data_explained
- [17] *Magnetometer - W3C Specification* <https://w3c.github.io/magnetometer/>
- [18] *Flask Web Framework* <https://flask.palletsprojects.com/en/2.0.x/>
- [19] *Repositorio propio de Github con la aplicación en Flask de BeCaptchaWeb.* <https://github.com/alrtfm/becaptchaweb>
- [20] *reCAPTCHA* — Google Developers <https://developers.google.com/recaptcha>